Index

Descripció del problema:

Pregunta 1: Map-Reduce - MongoDB

Partint del problema plantejat al primer lliurament i una llista de ciutats europees (amb les seves coordinades) es demana que implementeu el job que mitjançant un map reduce calculi el número d'avions que han sobrevolat cadascuna de les ciutats (considerarem una zona de 50Km de radi a partir de les coordinades donades).

La llista de ciutats per a realitzar el càlcul és la següent:

Ciutat	Latitud	Longitud
Amsterdam	52.371807	4.896029
Barcelona	41.390205	2.154007
Berlin	52.520008	13.404954
Dublin	53.350140	-6.266155
London	51.509865	-0.118092
Madrid	40.416775	-3.703790
Paris	48.864716	2.349014
Rome	41.906204	12.507516
Zurich	47.451542	8.564572

I l'objectiu és per cadascuna de les ciutats calcular el nombre d'avions diferents que les han sobrevolat amb les dades que tenim. També s'inclouen aquells avions que hagin aterrat a algun dels aeroports d'aquests ciutats, si estan dins del radi de 50Km des de les coordenades de la taula anterior.

Màquina virtual

Per tal de portar a terme la pràctica es proporciona una màquina virtual (Ubuntu Server 16.04 LTS 64 bits) preparada per Virtual Box. Trobareu un document junt amb aquest enunciat on s'explica com descarregar i utilitzar aquesta màquina virtual.

Joc de dades proporcionat

Juntament amb la màquina virtual es proporciona una bdd mongoDB ja carregada que conté aproximadament 2.000.000 de registres ADSB corresponents tots al continent europeu (latituds entre 36 i 61 nord, longituds entre 10 oest i 60 est) recollits mitjançant l'API streaming de ADSBExchange[1] durant unes hores el 17 de febrer de 2018.

La base de dades ocupa aproximdament 1Gb i conté una col·lecció anomenada «adsb» amb les dades ADS-B:

```
student@uoc:~$ mongo uoc
MongoDB shell version: 2.6.10
connecting to: uoc
> show dbs
admin (empty)
local 0.078GB
uoc 0.953GB
```

```
> show collections
adsb
system.indexes
> db.adsb.stats()
{
"ns" : "uoc.adsb",
"count" : 1998323,
"size" : 306795856,
"avgObjSize" : 153,
"storageSize" : 460894208,
"numExtents" : 15,
"nindexes" : 1,
"lastExtentSize" : 124993536,
"paddingFactor" : 1,
"systemFlags" : 1,
"userFlags" : 1,
"totalIndexSize" : 64852032,
"indexSizes" : {
"_id_" : 64852032
},
"ok" : 1
}
> db.adsb.findOne()
{
"_id" : ObjectId("5a8843e7d79a740f272ccc0a"),
"Vsi" : 832,
"Call" : "RYR8WE",
"Spd" : 229,
"Long" : 6.47644,
"Trak" : 1.3,
"Lat" : 51.530952,
"GAlt" : 4582,
"Sqk" : "2347",
"Icao" : "4CA9C4",
"ts" : ISODate("20180217T16:
01:59.173Z")
}
```

El significat dels camps[2] és el següent:

Camp	Descripció
`_id`	Identificador registre
`Vsi`	Velocitat vertical en peus per minut
`Call`	El «callsign» de l'avió.
`Spd`	Velocitat respecte terra en nusos
`Long`	Longitud
`Trak`	Direcció de l'avió (0 graus equival a nord)
`Lat`	Latitud
`GAlt`	Altitud en peus
`Sqk`	Codi «squack»
`Icao`	Codi ICAO (International Civil Aviation Organization) de l'avió
`ts`	Timestamp

Càlcul del map Reduce

Per a fer el map reduce haureu de seguir les següents passes:

1. En el map s'ha de fer un emit per cada registre ADS-B que estigui dins del radi d'una ciutat. S'haurà d'indicar el codi ICAO de l'avió que l'ha generat per després poder comptar els avions. Aquest codi ICAO és únic per aeronau.

2. En el reduce s'han d'agrupar els avions que han sobrevolat cada ciutat evitant així els duplicats (un avió haurà emès molts de registres ADS-B mentre sobrevolava una ciutat).

3. En el reduce comptem els avions diferents que han sobrevolat cada ciutat i donem el resultat final.

Per a la realització del map reduce es proporciona el script mapReduce.js amb l'estructura del mapreduce ja preparada. A més ja s'ha inclòs un array amb la llista de ciutats i les seves coordinades geogràfiques i la funció que permet calcular la distància entre dos jocs de coordinades.

Aquest script el podeu trobar al directori arrel de l'usuari student de la màquina virtual proporcionada. Heu d'implementar les tres funcions map, reduce i finalize, completant allà on s'indica un //TODO.

Exemple de mapReduce

Amb l'enunciat s'inclou un exemple de map reduce que calcula el número de missatges ADS-B segons el «flight level», és a dir, les franges d'alçada de 1000 peus en les que es divideix l'espai aeri. L'exemple el podreu trobar a la màquina virtual i adjunt a aquest enunciat.
A la funció map es fa un emit per cada flight level:

```
var m = function() {
if (this.GAlt > 10000 && this.GAlt < 45000) {
var fl = Math.round(this.GAlt / 1000) * 1000;
emit (fl, 1);
}
};
```

I a la funció reduce es compten els registres ADS-B per a cada flight level:

```
var r = function(key, values) {
// print("reduce key=" + key + " values=" + values);
return Array.sum(values);
};
```

Executem el map reduce:

```
student@uoc:~$ mongo uoc mapreduceexample.js
MongoDB shell version: 2.6.10
connecting to: uoc
```

I finalment podem veure el resultat:

```
student@uoc:~$ mongo uoc
MongoDB shell version: 2.6.10
connecting to: uoc
> db.result_example.find().sort({value:1})
{ "_id" : 36000, "value" : 88474 }
{ "_id" : 35000, "value" : 76898 }
{ "_id" : 37000, "value" : 70560 }
{ "_id" : 34000, "value" : 67528 }
{ "_id" : 33000, "value" : 58301 }
{ "_id" : 11000, "value" : 55246 }
{ "_id" : 31000, "value" : 53113 }
{ "_id" : 38000, "value" : 50382 }
{ "_id" : 32000, "value" : 50211 }
{ "_id" : 30000, "value" : 47850 }
{ "_id" : 12000, "value" : 47794 }
{ "_id" : 29000, "value" : 47742 }
{ "_id" : 28000, "value" : 46906 }
{ "_id" : 27000, "value" : 45655 }
{ "_id" : 13000, "value" : 45237 }
{ "_id" : 24000, "value" : 45100 }
{ "_id" : 26000, "value" : 44739 }
```

```
{ "_id" : 25000, "value" : 44610 }
{ "_id" : 21000, "value" : 44088 }
{ "_id" : 15000, "value" : 43913 }
Type "it" for more
```

Material de consulta

La documentació de MongoDB és molt complerta i ha de ser el vostre referent alhora de realitzar la pràctica:
Ús de javascript a MongoDB:
• https://docs.mongodb.org/manual/core/server-side-javascript/
MongoDB tutorial (Getting Started)
• https://docs.mongodb.org/getting-started/shell/
Explicació Map Reduce:
• https://docs.mongodb.org/manual/core/map-reduce/
• https://docs.mongodb.com/manual/reference/command/mapReduce/
Reviseu les seccions: “Requirements for the map function”, “Requirements for the reduce function” i “Requirements for the finalize function”
• https://docs.mongodb.org/manual/tutorial/troubleshoot-map-function/
• https://docs.mongodb.org/manual/tutorial/troubleshoot-reduce-function/
Exemples de Map Reduce
• https://docs.mongodb.org/manual/tutorial/map-reduce-examples/

Resposta:
S’adjunta el codi en el fitxer zip que s’entrega a registre d’avaluació.

En la funció map he aconseguit identificar els vols que han estat a menys de 50 Km de distància. A la funció reduce he identificat i retornat un array eliminant els ICAOs dels vols repetits. El que no he aconseguit es mostrar per a cada ciutat el número de vols.

Pregunta 2: Cassandra

Suposant que al final la eina escollida fos Cassandra, argumenta quin hauria de ser el nivell de consistència (Consistency level) en els següents casos:

• La escriptura de dades ADS-B recollides per la xarxa d'estacions.
• La lectura de dades al realitzar processos de map-reduce com el plantejat a la pregunta 1

Infrastructura disponible

Suposem que hem utilitzat Cassandra per a resoldre el problema plantejat al primer lliurament amb una configuració de 9 màquines distribuïdes en tres centres de càlcul utilitzant una estratègia NetworkTopologyStrategy:
• 3 màquines al centre de càlcul 1 – factor de replicació 3
• 3 màquines al centre de càlcul 2 – factor de replicació 3
• 3 màquines al centre de càlcul 3 – factor de replicació 3
A cada centre de càlcul els tres servidors es consideraran a racks diferents.

La consistència "ajustable" de Cassandra

El factor de replicació (replication factor) és el numero de vegades que cada dada es replica en un centre de càlcul (datacenter). El nivell de consistència sempre es defineix en base a aquest número, que ha de ser menor o igual que el número de màquines disponibles.
Cassandra disposa de consistència "ajustable". És a dir, permet establir nivells diferents de consistència a cadascuna de les operacions de forma diferent. Això equival a que l'usuari pot escollir quantes màquines del cluster contestaran a un determinat SELECT o qualsevol altra instrucció enviada a la base de dades. Als links del proper apartat podreu trobar els diferents nivells de consistència possibles explicats.

Material de consulta

Datastax és la principal empresa que hi ha darrera Cassandra i disposa de molta

documentació online. Recomanem les següents planes per a resoldre aquesta pregunta:
• https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlAboutDataConsistency.html
• https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html
• http://www.slideshare.net/DataStax/understanding-data-consistency-inapache-cassandra

Resposta:

La infraestructura serà de 9 màquines distribuïdes en tres centres de càlcul utilitzant una estratègia NetworkTopologyStrategy. Cada CPD disposa d'un factor de replicació de 3 i cada equip està en diferents racks. Un factor de replicació de 3 a un CPD significa que amb una consistència de nivell QUORUM, es permet que caigui un equip i consistència de nivell ONE. En el cas d'aquesta pràctica amb tres CPDs per a LOCAL_ONE, es necessari 1 node del LOCAL DC. LOCAL_QUORUM, necessita 2 nodes del LOCAL DC, EACH_QUORUM, necessita 2 nodes de cada EACH DC, en el nostre cas (6 nodes/equips).

Quin nivell de consistència assignaríem en els següents casos:

• La escriptura de dades ADS-B recollides per la xarxa d'estacions.

Cassandra permet els següents nivells de consistència per l'escriptura:
font: https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html

Write Consistency Levels

Level	Description	Usage
ALL	A write must be written to the commit log and memtable on all replica nodes in the cluster for that partition.	Provides the highest consistency and the lowest availability of any other level.
EACH_QUORUM	Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in *each* datacenter.	Used in multiple datacenter clusters to strictly maintain consistency at the same level in each datacenter. For example, choose this level if you want a read to fail when a datacenter is down and the QUORUM cannot be reached on that datacenter.
QUORUM	A write must be written to the commit log and memtable on a quorum of replica nodes across *all* datacenters.	Used in either single or multiple datacenter clusters to maintain strong consistency across the cluster. Use if you can tolerate some level of failure.
LOCAL_QUORUM	Strong consistency. A write must be written to the commit log and memtable on a quorum of replica nodes in the same datacenter as the coordinator. Avoids latency of inter-datacenter communication.	Used in multiple datacenter clusters with a rack-aware replica placement strategy, such as NetworkTopologyStrategy, and a properly configured snitch. Use to maintain consistency locally (within the single datacenter). Can be used with SimpleStrategy.
ONE	A write must be written to the commit log and memtable of at least one replica node.	Satisfies the needs of most users because consistency requirements are not stringent.
TWO	A write must be written to the commit log and memtable of at least two replica nodes.	Similar to ONE.
THREE	A write must be written to the commit log and memtable of at least three replica nodes.	Similar to TWO.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local datacenter.	In a multiple datacenter clusters, a consistency level of ONE is often desirable, but cross-DC traffic is not. LOCAL_ONE accomplishes this. For security and quality reasons, you can use this consistency level in an offline datacenter to prevent automatic connection to online nodes in other datacenters if an offline node goes down.
ANY	A write must be written to at least one node. If all replica nodes for the given partition key are down, the write can still succeed after a hinted handoff has been written. If all replica nodes are down at write time, an ANY write is not readable until the replica nodes for that partition have recovered.	Provides low latency and a guarantee that a write never fails. Delivers the lowest consistency and highest availability.

S'ha de tindre en compte que pot haver al mateix temps uns 15000 avions volant i enviant missatges cada 10 segons, que generaran un volum mig de 1500 missatges/segon a escriure a la base de dades. Considerem que l'escriptura es continuada a mida que es van rebent els missatges i degut a que el que es cerca es donar un servei de tracking en temps real i càlcul de horaris previst, les dades han d'estar sempre disponibles i el més actualitzades possibles, per tant es prioritza la disponibilitat de dades respecte a la velocitat d'escriptura i la seva consistència.

El nivell de consistència LOCAL_QUORUM amb la infraestructura proposada ens donara una tolerància a fallida perquè obtenir una alta disponibilitat, sense sacrificar consistència però sí que les estritures siguin més lentes perquè s'hauran de replicar a més nodes perquè estiguin disponibles per llegir. Aquest tipus de consistència s'utilitza en clústers on s'empra la estratègia NetworkTopologyStrategy , on amb 3 nodes a cada CPD permet la fallida d'un node per grup de replica, mantenint una alta consistència.

• La lectura de dades al realitzar processos de map-reduce com el plantejat a la pregunta 1

Els nivells de consistència de lectura que permet Cassandra son els mateixos que d'escriptura, excepte EACH_QUORUM i ANY però s'afegeixen el SERIAL i LOCAL_SERIAL.
font: https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlConfigConsistency.html

Read Consistency Levels

Level	Description	Usage
ALL	Returns the record after all replicas have responded. The read operation will fail if a replica does not respond.	Provides the highest consistency of all levels and the lowest availability of all levels.
EACH_QUORUM	Not supported for reads.	
QUORUM	Returns the record after a quorum of replicas from all datacenters has responded.	Used in either single or multiple datacenter clusters to maintain strong consistency across the cluster. Ensures strong consistency if you can tolerate some level of failure.
LOCAL_QUORUM	Returns the record after a quorum of replicas in the current datacenter as the coordinator has reported. Avoids latency of inter-datacenter communication.	Used in multiple datacenter clusters with a rack-aware replica placement strategy (NetworkTopologyStrategy) and a properly configured snitch. Fails when using SimpleStrategy .
ONE	Returns a response from the closest replica, as determined by the snitch. By default, a read repair runs in the background to make the other replicas consistent.	Provides the highest availability of all the levels if you can tolerate a comparatively high probability of stale data being read. The replicas contacted for reads may not always have the most recent write.
TWO	Returns the most recent data from two of the closest replicas.	Similar to ONE .
THREE	Returns the most recent data from three of the closest replicas.	Similar to TWO .
LOCAL_ONE	Returns a response from the closest replica in the local datacenter.	Same usage as described in the table about write consistency levels.
SERIAL	Allows reading the current (and possibly uncommitted) state of data without proposing a new addition or update. If a SERIAL read finds an uncommitted transaction in progress, it will commit the transaction as part of the read. Similar to QUORUM.	To read the latest value of a column after a user has invoked a lightweight transaction to write to the column, use SERIAL . Cassandra then checks the inflight lightweight transaction for updates and, if found, returns the latest data.
LOCAL_SERIAL	Same as SERIAL , but confined to the datacenter. Similar to LOCAL_QUORUM.	Used to achieve linearizable consistency for lightweight transactions.

Es considera que un servei de tracking en temps real i serveis de càlcul de rutes, hores previstes d'arribada entre d'altres ha donar sempre una resposta i el més ràpid possible, encara que la precisió no sigui la més exacta i els resultats s'actualitzin a mesura que arribin les dades. L'usuari no esperarà a que es mostrin resultats, i per tant es més important donar una previsió correcta amb les dades actuals i que s'actualitzi a mida que arribin les dades actualitzades. Per tant prioritzarem la disponibilitat i velocitat de lectura per sobre de la consciencia.

El nivell de consistència ONE permet la màxima disponibilitat sacrificant que les dades no siguin les últimes escrites.