Riak: analyseren van grote hoeveelheden data

Steven Raaijmakers

March 2017

1 Inleiding

Wanneer er grote hoeveelheden data beschikbaar zijn is het zaak deze goed te organiseren, zodat de data goed toegankelijk is voor gebruikers. Hierdoor ontstaat ook de mogelijkheid iets zinnigs te zeggen over de data. Bij het Enron fraude schandaal lekten de e-mails van een grote groep mensen uit. Omdat het hierbij in totaal over zo'n anderhalve GB aan data gaat, is het vrijwel onmogelijk om deze binnen afzienbare tijd handmatig te analyseren.

2 Probleemstelling

Door de e-mails te organiseren in een key-value-database kunnen de e-mails gemakkelijker geanalyseerd worden. Er kan bijvoorbeeld gezocht worden naar woorden die duiden op fraude zoals: "scam" of "cover". Het gebruik van bijvoorbeeld de Riak database kan dit proces versoepelen, omdat Riak de mogelijkheid biedt tot het gebruik van de mapreduce functie.

Om de data in een Riak database te zetten zal eerst de structuur van de e-mails moeten worden omgezet, zodat Riak deze accepteert.

3 Theorie

Riak is een gedistrubeerde NoSQL key-value database die bekend staat om zijn flexibiliteit. De daatabase biedt ook goede mogelijkheden voor schaalbaarheid en hoge fout-tolerantie. Riak is een opensource project, maar heeft ook een enterprise versie waarin opslag in de cloud mogelijk is.

3.1 Database initialiseren

De Riak database kan gedownload worden volgens de instructies in de sylabus. Wanneer deze eenmaal gedownload, is kan hij geïnstalleerd worden. Dit gaat door in de map van Riak het volgende aan te roepen:

```
make DEVRELL
```

Er zal nu een map ontstaan “dev” met daarin meerdere mappen van “dev1” tot en met “dev8”. Elke map staat hierbij voor één node. Verschillende nodes kunnen met elkaar *gejoined* worden tot één cluster. Stel dat node 1, node 2 en node 3 samen in een cluster moeten, kan dat gedaan worden door deze nodes eerst te starten:

```
dev/dev1/bin/Riak start
dev/dev2/bin/Riak start
dev/dev3/bin/Riak start
```

Nu de nodes eenmaal draaien kunnen ze elkaar joinen:

```
dev/dev2/bin/Riak-admin cluster join dev1
dev/dev3/bin/Riak-admin cluster join dev2
```

Hierbij joined node 2 node 1, en node 2 wordt gejoined door node 3, zodat er een groter cluster ontstaat.

4 Experimenten

4.1 e-mails naar Riak

Zie s.py

Eerst zijn de e-mails gedownload als .tar vanaf `https://www.cs.cmu.edu/~./enron/`. Daarna zullen de e-mails ingeladen moeten worden in Riak. Dit kan door middel van het schrijven van een script dat de e-mails aan Riak presenteert op een gangbare manier. Dit is gedaan door middel van de programmeertaal python, die een Riak module heeft, te downloaden via pip:

```
sudo pip install Riak
```

Vervolgens kan verbinding gemaakt worden met het cluster door te verbinden met één van de deelnemende nodes:

```
myClient = RiakClient(pb_port=10017, protocol='pbc')
```

Hierna worden de mappen uitgelezen uit de .tar file. Elke map die het script tegenkomt (van één diep) is een persoon. Elke persoon wordt in Riak opgeslagen als een *bucket*.

Vervolgens wordt gekeken naar alle mappen die een persoon heeft. Vaak zijn dit email mappen. Deze email-mappen worden op hun beurt weer uitgelezen, en als deze map een e-mail betreft, zal deze aan een dictionary worden toegevoegd. De e-mail map is in dit geval de *key* en de *value* is een *dictionary* van email-nummers naar email-tekst.

Een *entry* voor de bucket Slinger-S zal er dus als volgt uitzien:

```
inbox:
    {1: "lorem ipsum",
    2: "dolor"}
```

```
deleted:
    {1: "sit",
    2: "amet"}
```

Er ontstaan problemen zodra de er bijvoorbeeld een bestand gevonden wordt wanneer er een folder verwacht wordt. In dat geval wordt de file overgeslagen, waardoor er echter wel wat informatie verloren gaat. Dit is zelden het geval. Ook zijn er moeilijkheden wanneer de ingelezen e-mail niet volledig uit *utf-8* tekst bestaat. In dat geval zal de e-mail eerst *decoded* worden naar utf-8, alvorens deze in de Riak-database geplaatst wordt.

4.2 MapReduce

Nu de data eenmaal gestructureerd in de Riak DB gezet is, moet deze doorzocht worden. Dit doen we eerst lokaal, in de mappen. Dit gaat bijvoorbeeld door *grep*. Het doorzoeken van de e-mails van Swerzbin M. naar het woord "fraud":

```
grep -R -i "fraud" maildir/swerzbin-m/ | wc -l
```

Het eerste commando (voor de "pipe"), zoekt in alle bestanden naar het woord "fraud", het tweede commando telt vervolgens het aantal keer dat dit woord gevonden is.

Er kan ook gezocht worden naar hetzelfde woord in alle directories, door de laatste directory uit het path te halen:

```
grep -R -i "fraud" maildir/ | wc -l
```

Dit duurt echter zeer lang, het is daarom interessanter de mapreduce functie van Riak te gebruiken.

De mapreduce functie van Riak kan op verschillende manieren gebruikt worden. In dit geval zal dit via het versturen van een JSON-object van een bepaalde structuur door middel van het cURL-commando. De query uit de syllabus werkt helaas niet, dus werd de query herschreven. Het herschrijven gebeurde door middel van het gebruik van een shell script. Hierdoor ontstaat de mogelijkheid de queries te typen in een editor in plaats van in de commandline.

Op de volgende manier wordt er een POST-request gedaan naar het juiste address.

```
curl -X POST "http://localhost:10018/mapred" \
-H "content-type: application/json" -d {JSON_OBJECT_HIER}
```

Er wordt hier aangegeven dat er (naar een speciaal gedefinieerde url) een POST request wordt gestuurd met een JSON object.

Het JSON object moet de volgende structuur hebben:

```
{
    "input": $bucket,
    "query": [{
        "map": {
```

```
            "language": ...
            "source": ...
        }
    },{
        "reduce: {
            "language": ...
            "source": ...
        }
    }]
}
```

4.2.1 Map

Zie mr.sh

De mapreduce functie ziet er als volgt uit:

```
"map": {
       "language": "javascript",
       "source": "
       function(v) {
           v = v.values[0].data.toLowerCase().match(/'${WORD}'/g)
           var count = 0;
           (v || []).forEach(function(val) {
               count += 1;
           });
           return [count]
       }
    "
}
```

Er wordt gekeken naar de input die binnen komt v. Dit is een object met veelal andere soort data. Deze hoeveelheid wordt verkleind naar het deel dat we nodig hebben door uit dit object, de .values te pakken, hiervan het eerste item te nemen, en hiervan de .data op te vragen. Zo wordt v de inhoud van enkel de values. Vervolgens wordt al deze inhoud lowercase gemaakt, en gematched aan het opgegeven *$WORD*. Het resultaat van deze matching is een lijst met de daarin het *$WORD* in de hoeveelheid waarin deze voortkomt.

4.2.2 Reduce

Zie mr.sh

In de reduce functie worden alle hoeveelheden samen opgeteld tot één getal:

```
"reduce": {
    "language": "javascript",
    "source": "function(v, k){
        var sum = v.reduce(add, 0);
```

```
        function add(a, b) {
                return a + b;
        }
        return [sum];
    }"
}
```

v is in dit geval een lijst met de frequenties van het woord.

Het mapreduce-script is nu als volgt te runnen:

```
sh mr.sh \$PERSOON \$WOORD
```

Waarbij *$PERSOON* en *$WOORD* uiteraard vervangen moeten worden door respectievelijk de persoon waar op gefilterd moet worden, en het woord waarnaar er binnen deze persoons bestanden moet worden gezocht.

4.2.3 Meerdere buckets

Zie all.py

In de documentatie van Riak staat genoteerd dat de mapreduce functie slechts één bucket per keer toe laat. Daarom heb ik de mogelijk toegevoegd om het resultaat van *mr.sh* bij elkaar te nemen. In all.py worden eerst alle buckets opgevraagd uit de Riak database, hier wordt vervolgens een lijst van gemaakt.

Voor elk element uit de bucket-lijst wordt het shellscript van *mr.sh* uitgevoerd. Deze resultaten worden door python bij elkaar gevoegd en opgeteld. Zo kan er gezocht worden in meerdere buckets, maar hierdoor verliest de mapreduce wel een groot gedeelte van zijn kracht, omdat de mapreduce slechts per bucket gedaan wordt.

5 Resultaat

Het script dat e-mails inlaadt lijkt redelijk vlot te zijn, als gekeken wordt per email die ingelezen wordt een naar de database geschreven wordt. In het script wordt alleen geprint wanneer een gehele persoon/bucket is toegevoegd. Dus wanneer een persoon veel verschillende e-mails heeft om te verwerken, lijkt het script te vertragen, maar dit is niet het geval.

In totaal doet het script ongeveer een kwartier over het inladen van de email-database, die zo'n anderhalve GB groot is.

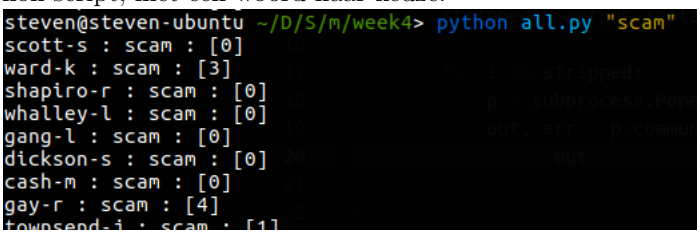
5.1 Mapreduce

De grep functie naar de frequentie van het woord "fraud" in de bestanden van Swerzbin M leverde "1" op.

Het runnen van het mapreduce script levert dezelfde waarde op:

```
steven@steven-ubuntu ~/D/S/m/week4> sh mr.sh swerzbin-m fraud
swerzbin-m : fraud : [1]⏎
```

De mapreduce over meerdere buckets gaat door het aanroepen van het all.py python script, met een woord naar keuze:



Deze waardes kunnen vervolgens opgeteld worden om zo de totale frequentie te berekenen.

6 Conclusie en Discussie

De mapreduce funcite over alle buckets verloopt veel sneller dan de lokale grep. Toch neemt het wel wat tijd in beslag. Dit zou kunnen worden verholpen als de mapreduce zou werken op meerder buckets, wat niet het geval is. Een andere mogelijkheid is de structuur zo te veranderen dat alle e-mails samen in één bucket geplaatst kunnen worden. De syllabus adivseerde echter om per persoon een bucket te maken, wat op zich logisch klinkt. Het is daarom jammer dat Riak geen ondersteuning biedt voor een mapreduce op meerdere buckes.

Hoewel Riak claimed een hoge fouttolerantie te hebben, heb ik er toch veel problemen mee gehad. Het downloaden van Riak verloopt alleen soepel als je de de juister Erang versie hebt, wat een gedoe an sich is. Wanneer dit eenmaal gelukt is verloopt het plaatsen van de data redelijk soepel, alleen wil Riak dat alles in utf-8 gecodeerd is.

De gebruikte interface voor Riak is Rekon, waardoor je via de UI toch kan checken of de data goed geplaatst wordt. Je kan hier echter weinig aan de instellingen doen van Riak. Die zullen toch echt verandert moeten worden door in de .config files te kijken en te hopen dat je de goede variabele aanpast.