Portfolio assignment - part one

1 Introduction

The portfolio in this course will be individual and consists of two projects, of which, this is the first part. In this part of your portfolio, you will use data from OpenStreetMap (OSM) to show that you understand the concept of Big Data and how the Hadoop Distributed File System (HDFS), MapReduce and Apache Spark works.

2 Primary data source

The primary source of data for your project will be the OpenStreetMap project. OpenStreetMap is a collaborative project where volunteers around the world draw a map from sources such as arial imagery, GPS devices, local knowledge and more. For our purposes, it's more important to think of not what you see when you look at the map on OpenStreetMap.org, but rather what that map visualizes. That is, the database containing geographical information.

In terms of Big Data use cases, this is likely one of the most general, and therefore important, sources of data available for free. This is because whether you are designing a new system for predicting the sales of candy, improving the logistics of a retail store chain, or trying to determine the optimal place to put a new bicycle lending station, having digitized knowledge of the world around you is essential.

For this project you will use the OSM XML files that are provided by mirrors of the database. The easiest solution to retrieve the data is to download an extract of your choice by using the site https://extract.bbbike.org/ and selecting either the OSM XML 7Z (xz) format or the OSM XML gzip'd format. Larger extracts can be downloaded from https://download.geofabrik.de/ index.html.

I recommend downloading only a small region, such as a city. When uncom- pressed, the files can become rather large, only downloading a region containing Fredrikstad, Sarpsborg and Halden can easily be 230 MB when uncompressed. Being able to open the file in a text editor will greatly ease your work.

A documentation of the OSM XML file format is available here:
https://wiki.openstreetmap.org/wiki/OSM_XML

Jeg har valgt å velge Oslo og omegn.
Jeg lastet tidlig ned en osm-fil som viste seg å bli altfor stor. Senere i prosjektet endte jeg derfor opp med å hente en ny, med en mindre del av Oslo.
Informasjon om data som er hentet:

Name: oslo
Coordinates: 10.73,59.918 x 10.759,59.933
Script URL:
https://extract.bbbike.org?sw_lng=10.73&sw_lat=59.918&ne_lng=10.759&ne_lat=59.933&format=osm.gz&city=oslo&lang=en
Square kilometre: 2
Granularity: 100 (1.1 cm)
Format: osm.gz
File size: 0.6 MB
SHA256 checksum: 7093b53dcaacacf9e37c8df93ffdd20e7040a57d3aaa8458aed808caf3a9ea33
MD5 checksum: 49bde295af7b6c6af451857c9d5c1c46
Last planet.osm database update: Thu Sep 26 16:22:32 2019 UTC
License: OpenStreetMap License

3 Assignment parts

3.1 Method for importing initial data into HDFS

Oppgavetekst

Describe how you import the data into HDFS, and what happens "behind the scenes" when you write the data to the file system. Additionally, describe what will be different between your setup and a cluster consisting of multiple machines and when HDFS is configured to use multiple replicas of the file.

Importere data

Det er flere måter å importere data inn til HDFS. For å lese/skrive data til HDFS kan man benytte seg av verktøy som Sqoop, Flume, Hadoop Distcp, Bruke HUE, File browser eller Benytte et JAVA API. Jeg har valgt å gjøre dette mer manuelt ved å benytte meg av Hadoop File system sine kommandoer. Jeg utførte følgende:

1. Starter Hadoop: Start-all.sh
2. Sjekker at Hadoop er oppe og kjører: jps
3. Legger fila inn i filsystemet: Hdfs dfs -put navnpåfil.osm /navnpåfil.osm
    a. Eventuelt hadoop fs -copyFromLocal navnPåfil.som /navnpåfil.osm
4. Ser at filen er lagt i hadoop sitt filsystem: hdfs dfs -ls

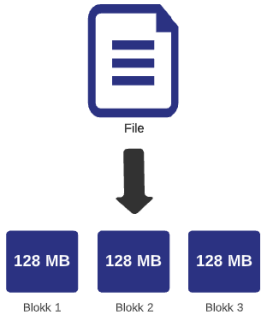
Hva skjer i bakgrunnen:

Litt HDFS-arkitektur
For å bedre kunne forklare hva som skjer ønsker jeg først å forklare litt om de ulike komponentene i Hadoop Distributed File System (HDFS).

Med HDFS bli data distribuert over flere ulike maskiner. Alle disse maskinene er koblet sammen og utgjør sammen et såkalt *Hadoop Cluster.* Hvert cluster består av, blant annet, en NameNode som har en rolle innenfor YARN-rammeverket og dermed håndterer filsystemet og har oversikt over ledige DataNoder, en Secondary nameNode som fungerer som avlastning og backup for NameNoden, DataNoder hvor selve dataen blir lagret og en ClientNode som styrer kommunikasjonen mellom NameNoden og DataNodene,

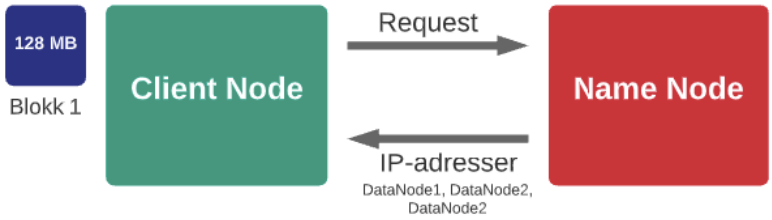
<u>Skrive data til HDFS</u>

Når filen skal skrives til HDFS deles denne først opp i blokker på 128 MB (Merk at størrelsen på disse blokkene kan variere noe).



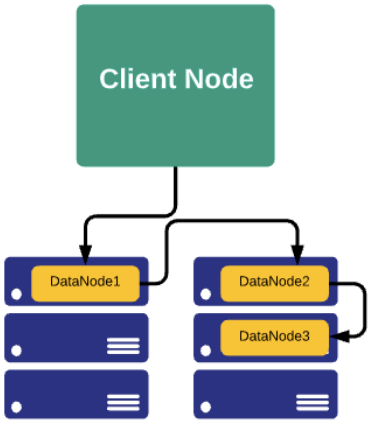
Deretter skjer følgende steg samtidig for hvert block:

1. Pipeline Setup
   I første steg kontakter ClientNoden Namenoden og gir beskjed om at den har en Block den trenger å lagre (Sender en request). NameNoden responderer med å returnere en liste med IP-adresser til 3 ulike DataNoder, da standard for antall kopier av en Block er 3.

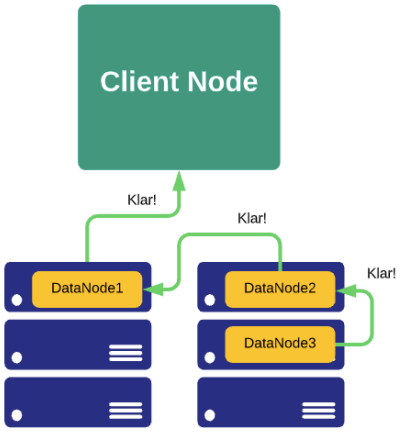


   Årsaken til disse kopiene er for å hindre at data blir tapt om en datanode eventuelt skulle sluttet å fungere.
   Den første blocken vil lagres på den lokale maskinen, den andre på en annen rack (som består av flere DataNoder) og den tredje skal lagres på en annen DataNode på denne racken. Det vil maks være en duplikat per DataNode og kun to per rack.

   Når ClientNoden får denne listen returnert vil den kontakte den første DataNoden for å sikre at denne er klar til å ta imot data. Samtidig gir den beskjed om å kontakte den andre DataNoden, som igjen kontakter denne tredje for å sørge for at denne er klar til å ta imot data.



Skulle en av disse ikke gi et klarsignal, vil ClientNoden gå tilbake til NameNoden og be om IP-adresse til en ny DataNode. NameNode sjekker så hvilke DataNoder som er ledige og velger deretter ut en av de som har sendt såkalte «Heartbeats»-signaler om at den er ok, for å returnere tilbake IP-adressen til denne tilbake ClienNoden.
Når alle DataNodene 3 gir et OK-signal er pipelinen satt opp og hdfs er nå klar til å lagre dataen.



2. Lagring av Data

   På samme måte som kontrollen av DataNodene skjedde vil også lagringen foregå. ClientNoden kontakter den første for å lagre, den første kontakter den andre og den andre kontakter den tredje.

3. Bekreftelse på lagring - Acknowledge

   Når den siste DataNoden har lagret den tredje kopien vil den sende en såkalt ACK-melding (Acknowledge) til den forrige DataNoden for å gi beskjed om lagringen er gjort uten problemer. Slik fortsetter prosessen bakover til ACK-meldingen har nådd ClientNoden og denne får beskjed om at alle 3 Blockene er lagret.
   Denne beskjeden videreføres til NameNoden slik at denne kan oppdatere sin informasjon om de ulike DataNodene.

   Kilde:

   https://www.youtube.com/watch?v=mafw2-CVYnA&fbclid=IwAR1mstik2Vsg2rHdlkMDerC5eMN-

od1MwmbuHUh2h8JOCkLZryzUGn8K1_E

Forskjellen på min setup, et cluster som består av flere maskiner og når HDFS er konfigurert til å bruke flere duplikater av filen.

Jeg har vanskeligheter med å forstå dette spørsmålet. Ønsker tilbakemelding på om jeg tenker korrekt.

Med et cluster vil data bli distribuert utover flere maskiner. Med cluster vil det være enkelt å skalere opp eller ned, da det kun er å legge til eller trekke fra maskiner til clusteret. Da flere maskiner som kjører minsker dette sjansen for feil, ettersom andre kan ta over om en a de skulle feile. Setup med cluster vil derfor være ganske fleksibelt.

Om HDFS er konfigurert til å bruke flere duplikater (replicas) av filen vil alt være distribuert utover på en maskin, og ulike deler av filen vil bli brukt for å parallell prosessering.
Det fine med Hadoop er at dette ikke er noe vi egentlig trenger å tenke så mye på hvordan den kjører da dette blir abstrahert for oss. Men som sagt blir hovedforskjellen hvor mye data vi faktisk kan håndtere.

3.2 Method for updating the data

OpenStreetMap is continually updated. Design an approach for updating the data stored in HDFS with updated data. Any working approach will be accepted, but bonus points will be given to solutions which also reduces the load on the cluster and prevents the data from becoming unavailable to jobs which may already be running.

Jeg er noe usikker på om jeg svarer korrekt på denne oppgaven. Ønsker derfor tilbake melding også på denne om jeg er i nærheten.

Oppdatere data I HDFS

Å oppdatere data i HDFS kan være utforende da dette filsystemet er «immutable», altså uforanderlig. Filer som er lagt til filsystemet kan ikke endres. Samtidig finnes det heller ingen oppdaterings-kommandoer man enkelt kan benytte seg av. Likevel er det noen metoder man kan ta i bruk for å få gjort endringer. Jeg vil forklare fire ulike metoder. Hybrid Update, HBase Dimension Store og Merge and Compact Update (med «Good Enough» Update).

Hybrid Update

Hybrid Update-metoden benytter ETL (Extract Transform and Load) og SQL-programmering. Ved hjelp av et verktøt kalt Sqoop vil all data bli kopiert med jevne mellomrom. Dette er ofte den første som blir tatt i bruk da den er såpass enkel å implementere.

Bakdeler med Hybrid Update er at denne metoden ikke er skalerbar og dermed ikke særlig effektiv ved store datamengder, da all alle record-ene må hentes hver gang en oppdateirng skal gjennomføres.

HBase Dimension Store

HBase er en NoSql-database som kjører på hadoop og har med dette innebygde verktøy for oppdatering av data. Denne metoden fungerer best om det kun er en eller et fåtall rader som skal modifiseres. Kreves det at hele tabellen må gås igjennom er HBase lite effektiv.

Merge And Compact Update

I motsetning til metodene over benytter denne seg av algoritmer for å oppdatere data. Her brukes altså innen annen spesifikk teknologi. Denne algoritmen kan implementeres med blandt annet både Java MapReduce og Spark. Algoritmen går igjennom seks steg:

1. **Kopier data:** En full kopi av dataen lagres i HDFS (Denne kopien blir ofte kalt «Masterdataen»).
2. **Laste inn ny data:** Den nye oppdaterte dataen blir deretter lastet inn i HDFS (Ofte kalt «Deltadata»)
3. **Flett data sammen:** Neste steg flettes den kopierte og oppdaterte sammen etter nøkkelfeltet.
4. **Plukk ut data:** Når man har kommet til denne delen vil man ha flere recorder for hver nøkkel. I denne delen vil algoritmen sørge for at det kun er en per nøkkel. Hvilken record som bli spart velges oftes utifra timestamp (Den som er sist endret).
5. **Skrive data midlertidig:** Resultatet av forrige steg vil nå bli skrevet til en midlertidig output (Da de fleste Hadoop-jobber ikke kan overskrive andre mapper).
6. **Overskriv masterdataen:** I dette siste steget vil dataen som ble midlertidig lagret i forrige flyttes til der masterdataen er lagret, og på den måten overskrive denne dataen.

Bakdelen med denne metoden er at man må lese og prosessere all data hver gang noe skal oppdateres, noe som er lite effektivt. Dette kan løses med noe kalt «Good Enough» Update

Denne metoden baserer seg på at noen data har større sjanse for å måtte bli oppdatert enn andre. Ved å flytte disse dataene (ofte de som sist ble oppdatert) i til egne sub-folders, vil Merge and Compact Update-metoden kun utføres på dataene som har størst sjanse for å skulle endres. På den måten slipper man å lese og prosessere alle dataene, og også ende opp med overflødige recorder

Kilde:

https://community.hitachivantara.com/s/article/hadoop-how-to-update-without-update

3.3 MapReduce and Spark

Unless otherwise agreed, the program submissions should use Java or Scala.

3.3.1 Calculate simple metrics

Write programs which answer the following questions about the OSM data. These are listed in an approximate order of increasing difficulty.

→Should be implemented as both MapReduce and Apache Spark programs

- *How many buildings is it in the extract you selected?* → ***One_BuildingCount***
- *How many addr:street tags exist for each street?* → ***Two_AddrStreetTagsPerStreet***
- *Which object in the extract has been updated the most times, and what object is that?* → ***Three_ObjectMostUpdated***
- *Which 20 highways contains the most nodes?* → ***Four_20TopHighWayNodes – Mangler sortering***
- *What is the average number of nodes used to form the building ways in the extract?* → ***Five_AverageNumOfNodesBuilding***
- *How many ways of types ”highway=path”, ”highway=service”, ”high- way=road”, ”highway=unclassified” contains a node with the tag ”bar- rier=lift gate”?*
- *Which 15 highways contains the most number of traffic calming=hump?*
- *Which building has the largest latitudinal extent? (biggest difference between the northernmost and southernmost node)*
- *What is the longest way with tag highway?*

Also see section 4.2 for expectations from the report.

3.3.2 Creative part

→ Should be implemented as both MapReduce and Apache Spark programs

In Big Data we often want to combine data from multiple sources. You will use one or more additional data sources of your choice in order to answer questions you ask yourself. You are free to use almost any data source which is either public, or for which you can provide the data. You are not required to use the same extract of OSM data as you used when calculating the simpler metrics. The only requirement is that the questions you define and write the programs to answer require the use of both the OSM data and your own data.

Some available data sources:
• https://oslobysykkel.no/apne-data
• https://kartkatalog.miljodirektoratet.no/Dataset • https://open.stavanger.kommune.no/dataset
• https://hub-frstadkomm.opendata.arcgis.com/

For the report, in addition to following the expectations for the report as outlined in section 4.2, you should also describe what your questions are, and why MapReduce or Spark is the correct or not the correct tool to answer them.

3.3.3 Compare the performance of the MapReduce implementation and the Spark implementation

All programs to the questions sections 3.3.1 and 3.3.2 should be implemented as both MapReduce and Apache Spark programs. You should then measure the performance when running both versions and explain what causes the differences.

4 Delivery

The delivery should be a zip file containing the following parts.

4.1 Source code

You should provide provide the source code you have used in this project as part of the zip file using a folder and naming scheme which clearly identifies the purpose of each file.

4.2 Report

You should provide the report for this project as a single PDF file. The contents of the report are expected to be fairly extensive(between 20 and (maximum) 50 pages). In addition to documenting what your programs do, you should also explain what happens ”behind the scenes” when you submit your program and during execution.

You should also explain what would happen behind the scenes if the programs were to be run on a large cluster consisting of many nodes and using a much larger extract of OSM data. Will your programs still produce the same results? If the results differ, are they still correct?

4.3 Data (if applicable)

If you use data for the ”creative part” of the project which is not available online, you should provide the data with your submission.