NTNU
Kunnskap for en bedre verder

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4305 Big Data-Arkitektur

Faglig kontakt under eksamen: Kjetil Nørvåg/Heri Ramampiaro
Tlf.: 73596755/73591459

Eksamensdato: 28. mai 2016
Eksamenstid (fra-til): 09.00-13.00
Hjelpemiddelkode/Tillatte hjelpemidler: D: Ingen trykte eller håndskrevne hjelpemiddel tillatt.
Bestemt, enkel kalkulator tillatt. Annen informasjon:

Målform/språk: Bokmål
Antall sider (uten forside): 4
Antall sider vedlegg: 0

Kontrollert av:

_____
Dato Sign

Informasjon om trykking av eksamensoppgave
Originalen er:
1-sidig X 2-sidig □
sort/hvit X farger □

_____ Merk!
Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Oppgave 1 – Big Data – 5 %

a) Når man skal forklare Big Data snakker man ofte om de tre (eller flere) V'ene. Forklar de tre viktigste av disse.

Volume: refers to the amount of data being generated and that needs handling (such as storage and analysis). There are several factors contributing to the large increase in data volume in the Big Data era versus previously, the most important being the growth of social media platforms, and the utilization of sensors. Both social media (uploading high resolution images and videos) and sensors (internet of things, credit card scanners, other autonomous systems) generate enormous amounts of data.

Velocity: refers to the speed at which data is being generated, accumulated, ingested, and processed. E. g Twitter users generate millions of tweets every minute that need to be stored and processed, for example to analyze trends etc. Utilization of social media and sensors is therefore also a big factor in explaining the increase of velocity in the big data era compared to previously.

Variety: refers to the variety in the data being generated. Before, it was a lot more common that a datasource only generated structured data, while now, datasources can generate combinations of structured, semi-structured, and unstructured data. Previously, sources of data were mainly transactions involving financial, insurance, travel, healthcare, retail industries, and governmental and judicial processing. Now, these sources have expanded dramatically, and includes satellite data, internet data (clickstream and social media), research data, images, videos, emails, signal data (sensors).

Oppgave 2 – Hadoop – 20 % (alle deler teller likt)

a) Hva var viktige mål for Hadoop File system (HDFS), og hva er HDFS *ikke* egnet til?

De viktigste målene for HDFS var følgende:

1. Bruk av veldig store filer. Tanken bak dette var å redusere total tid brukt på aksess. Ved å bruke store filer kan man finne entrypointet for informasjonen man leter etter, og lese kontinuerlig i den store fila før man trenger å aksessere et nytt entrypoint.
2. Datastrøm-aksess. Det at man ønsker datastrøm-aksess er også litt av grunnen til at man lagrer data i veldig store blokker (filer). I big-data sammenheng er datastrømmer et viktig aspekt, og målet med datastrøm-aksess i hadoop er at man skal lagre store filer, aksessere fila i begynnelsen, og leser store deler av fila kontinuerlig fra begynnelsen og utover. Ved å bruke store filer trenger man mindre søking enn dersom man heller bruker mange små filer.
3. Bruk av hyllevare-maskiner. Dette er fordi hyllevaremaskiner ofte er billige, og det gjør

systemet skalerbart ved at man kan kjøpe flere maskiner og koble de opp mot systemet ved behov for større lagringsplass, eller eventuelt fjerne maskiner og selge dem eller flytte dem dersom behovet for lagringsplass minker.

HDFS er ikke egnet til

1. Data-aksess av mindre datablokker med krav til liten forsinking. Dette fordi dataen ligger lagra i store filer, og systemet er lagd for aksessmønsteret at man aksesserer entry-pointet til ei fil og leser kontinuerlig store deler av fila.
2. Mange små filer. Fordi dataen er lagra i store filer som beskrevet over, og også på grunn av måten dataen aksesseres på. Navnenoder i HDFS holder all oppslagsinfromasjon om hver fil i hovedminnet, noe som kan bli vanskelig å få til dersom man bruker mange store filer (tar for mye plass til å passe i hovedminnet)
3. Flere klienter som skriver til ei fil simultant. Dette går ikke i HDFS, mest på grunn av at man ønsker å gjøre ting enklest mulig for å forbedre feiltoleranse og feilhåndtering. Altså kan kun en klient skrive til ei fil på en gang, men flere kan lese fra samme fil på samme tid.
4. Vilkårlige oppdateringer i filer. I HDFS er det ikke mulig å oppdatere ei fil på andre måter enn å legge til data på slutten av fila. Dvs at HDFS er append-only. Dersom man vil gjøre endringer på en fil på andre måter enn å legge til noe på slutten, må man gjøre dette ved å kopiere innholdet i fila, gjøre endringen, og så opprette ei ny fil og skrive den oppdaterte fila til den nye. Dette er svært lite effektivt.

b) Beskriv arkitekturen til HDFS (bruk gjerne figur). Beskriv hvordan filer er lagret og node-typer.

HDFS består av racks som igjen består av et visst antall hyllevare-maskiner. Innad i et rack, går kommunikasjonen mellom maskiner gjennom en switch som ligger i racket. Det vil si at hvis en node (maskin) i et rack vil kommunisere med en annen node i samme rack, så sender den først informasjonen til switchen, som sender det videre til den aktuelle noden. Man sier at nettverksdistansen for kommunikasjon mellom to noder i samme rack er 2 (1 for første kommunikasjon fra sender-noden til switchen, og 1 for kommunikasjon fra switchen til

mottakernoden). Nettverksdistansen her er svært kort, som gjør kommunikasjon mellom maskiner i samme rack veldig effektiv.

I tilleg til dette kan racks kommunisere med andre racks i systemet. Denne kommunikasjonen går gjennom en annen switch som kobler racksene sammen. Nettverksdistansen mellom to noder i forskjellige racks blir da 4 (først 1 for kommunikasjon mellom sender-node og intern rack-switch, så 1 for kommunikasjon fra intern rack-switch til switchen som kobler rackene sammen, så 1 for kommunikasjon fra den switchen til den interne rack-switchen som mottakernoden ligger i, og til slutt 1 for kommunikasjon fra den switchen til mottakernoden).

Disse rackene kan også være lokalisert i forskjellige datasenter (som kan være i forskjellige bygninger, eller forskjellige geografiske lokasjoner). Da får man enda en switch som sørger for kommunikasjon mellom datasenter. Nettverksdistansen mellom to noder i forksjellige datasenter blir 6 (først 1 fra sender-noden til intern rack-switch, så 1 fra intern rack-switch til switch som kobler rackene sammen, så 1 fra den switchen til switchen som kobler datasentrene sammen, så 1 fra den switchen til switchen som kobler rackene i mottaker-datasenteret sammen, så 1 fra den switchen til intern rack-switch i mottaker-racket, og til slutt 1 fra intern rack-switch til mottakernoden).

Datablokkene er lagret på store filer som ligger på disse maskinene. I HDFS bruker man partisjonering og replikering. Partisjonering betyr at deler av ei fil kan være lagra på forskjellige maskiner. For eksempel for ei fil som består av datablokker A, B og C, så kan A og B være lagra på samme maskin, mens C kan ligge på en annen maskin. Replikering betyr at hver datablokk er lagra flere steder, som oftes på tre forskjellige maskiner (tre-veis replikering). Dette øker feiltoleransen, ved at dersom en maskin krasjer, så ligger fortsatt de filene som eksisterte på den maskina også på andre tilgjengelige maskiner som kan håndtere forskjellige forespørsler.

Ettersom HDFS bruker replikering, så må man ha en node som kjører en tjeneste som kan fortelle hvor de forskjellige filene ligger lagret. Denne kalles Namenode (navnenode), og kan ta inn et filnavn og svare på blant annet hvor mange replikasjoner det finnes av den fila, og hvor de ulike

replikasjonene ligger (ved bruk av blokk-identifikatorene som den aktuelle fila består av, samt info om hvilke datanoder blokkene ligger på). Navnenoda holder all informasjon i hovedlageret for å sørge for effektivitet, og bruker kun disken til back-up. Det at all informasjon i navnenoda skal kunne ligge i hovedlageret er også en god grunn til å ha store filer, ettersom hver fil vil bruke plass på navnenoda, og med færre filer så blir det mindre plass brukt på navnenoda. Back upen som er lagret på disk i navnenoda består av et namespace-image (en kopi av det som ligger i hovedlageret på et visst tidspunkt), og en edit-log (en log som består av alle oppdateringer som gjøres i hovedlageret), og etter en viss tid, vil namespace-image og edit-log merges sammen, slik at man igjen blir up-to-date med det som ligger i hovedlageret.

Selve dataen ligger lagra i datanoder. En datanode lagrer hver datablokk som ei fil, og har blokk-identifikatoren som del av filnavnene. I tillegg til selve datablokkene, så inneholder datanoda også metadata-informasjon om hver av blokkene som ligger på den noda. Den viktigste metadata-informasjonen er en sjekksum for hver blokk. Den fungerer sånn at når en klient skriver til ei fil, så blir det også lagd en sjekksum som lagres som meta-data i den aktuelle datanoda. Når den samme fila skal leses igjen, så sjekker regner datanoda ut sjekksummen for det som ligger i fila og sammenligner med sjekksummen den har lagra i metadataen. Dersom de stemmer overens, så kan fillesingen fortsette, men hvis ikke så har det skjedd noe galt med fila, og man må da lese fila fra en annen replikasjon.

c) Forklar hva som skjer når en klient skal lese en fil som er lagret i HDFS (inkl. interaksjon mellom noder).

Når en klient skal lese en fil som er lagret i HDFS, sender klienten først en forespørsel med filnavn til navnenoden ved bruk av et Hadoop API. Navnenoden vil da slå opp i katalogen den har liggende i hovedminnet, og returnere informasjon om fila til klienten. Denne informasjonen inkluderer blant annet blokkidentifikatorene til blokkene som fila ligger lagra på, samt hvilke datanoder de blokkene ligger på.

Når klienten mottar denne informasjonen, vil den sende leseforespørselen direkte til den/de aktuelle datanoden(e). Datanoden som mottar forespørselen vil da først sjekke at sjekksummen med den som ligger i blokka som skal leses stemmer over ens med den sjekksummen den har liggende i meta-data-fila for den blokka, og dersom det stemmer, så begynner datanoden å sende over dataen i fila direkte til

klienten.

Oppgave 3 – MapReduce og Spark– 10 % (alle deler teller likt)

Anta at man har en fil PersonInfo.txt som inneholder informasjon om navn, alder og lønn, dvs. format som dette:

Kari 45 450000
Ola 30 200000
Kate 30 500000
Pål 45 550000

Vi ønsker å finne gjennomsnittsinntekt for hver alder, dvs. resultat som dette (trenger ikke å være sortert):

45 500000
30 350000

a) Vis med pseudokode for *mapper* og *reducer* hvordan dette kan gjøres i MapReduce. Anta for enkelhet skyld at *value* til map er en post med feltene *age* og *salary*, dvs. bruk følgende som utgangspunkt:

```
public void map(key(name), value(age,salary))
public void reduce(key, Iterable values)
```

```
public void map(key(name), value(age, salary)){
     skriv value[age], value[salary]
}

public void reduce(key, Iterable values){
     n = 0
     total = 0
     for each val in values:
          n++
          total += val
     avg_salary = total/n
     skriv (key, avg_salary)
```

```
}
```

b) Vi ønsker nå å finne maks-lønn for hver alder ved hjelp av Spark. Anta at vi allerede har lest filen inn i en RDD av par (key,value)=(age,salary), dvs. RDD[(int,int)]. Vis hvilke(n) transformasjon(er) som må gjøres for å få en resulterende RDD der (key,value)=(age,maxSalary). Hint: viktige transformasjoner og handlinger ("actions") i Spark inkluderer `map, flatMap, filter, distinct, union, collect, count, countByValue, reduce, reduceByKey, groupByKey, values, sortByKey,` og `countByKey.`

```
val maxSalary = rdd.reduceByKey((a,b) => Math.max(a,b))
```

Oppgave 5 – Datastrømmer (streaming data) – 30 % (Alle deler teller likt)

Du skal analysere hvor mange ganger et tema om amerikansk valg og valgkamp blir nevnt i meldinger i sosiale media som Twitter.

a) Drøft karakteristikkene og/eller utfordringene med datastrøm. Nevn to andre eksempler hvor man er nødt til å håndtere en datastrøm (i tillegg til Twitter og eller sosiale media generelt).

Karakteristikker:

- Datastrømmer er kontinuerlige strømmer av data.
- Det er snakk om store mengder data
- Man vet ikke når, eller om de slutter. Kan være uendelige.

Utfordringer:

Den største utfordringen med datastrømmer er at den endrer seg raskt og kontinuerlig, og derfor trenger rask respons i sanntid.

Eksempler hvor man må håndtere datastrømmer:

- Signaldata, for eksempel sensordata fra sensorer i en bil for å gjøre beregninger for å unngå kollisjoner
- Transaksjonsdata, for å detektere om deler av systemet er nede som kan føre til for eksempel tapt inntekt.

b) Vi skiller mellom to typer spørringer når det gjelder datastrøm. Forklar hva disse er. Bruk eksempler til å støtte forklaringen din.

Når det gjelder datastrøm skiller vi mellom to typer spørringer:

- Ad-hoc spørringer: disse spørres kun en gang. Et eksempel er hvis man har en strøm av kjøpstransaksjoner og ønsker å vite prisen for det dyreste kjøpet på slutten av en dag. Da kan man spørre hvilken verdi som har vært den høyeste i løpet av dagen.

- Stående sørringer: disse spørres kontinuerlig. Et eksempel er hvis man har en strøm av twitter-tags, og alltid har lyst til å vite hvilken tag som er mest brukt for å kunne analysere trender i sanntid. Da kan man feks ha en stående spørring "hvilken tag er mest brukt i dag", som vil kunne gi ulike svar til ulike tidspunkter på dagen.

c) Se for deg at du skal finne ut hvor stor andel av meldingene er relatert til temaet "valg" og "valgkamp" i en gitt begrenset tidsperiode. Til dette formålet velger vi å bruke glidende-vindu prinsippet ("sliding window"). Anta at dette vinduet har en størrelse på 1000 twitter-meldinger (dvs. Tweets). Vis hvordan du går fram for å beregne denne andelen.

For de første 1000 twitter-meldingene i vinduet, kan man beregne andelen ved å gå gjennom alle meldingene i vinduet, finne ut hvor mange som inneholder ordene "valg" og/eller "valgkamp", og dele dette på 1000 (antall meldinger totalt i vinduet).

Deretter må man hver gang den kommer en ny tweet inn i vinduet sjekke om denne meldingen inneholder de aktuelle ordene. Hvis den gjør det, så endrer man snittet med (1-j)/N hvor N er antall tweets i vinduet (1000), og j =1 dersom den eldste tweeten i vinduet inneholdt ordene, og j = 0 dersom den eldste tweeten i vinduet ikke inneholdt ordene.

d) Kan problemet over sees på som en variant av "bit counting"? Begrunn svaret ditt.

Ja, man kan representere tweets som inneholder ordene man leter etter som 1 og tweets som ikke gjør det som 0. Deretter kan man telle antall enere i et gitt størrelse av et vindu osv.

e) Bruk "bloom filter"-prinsippet til å fylle ut tabellen nedenfor

Strømelement	**Hash-funksjon - h_1**	**Hash-funksjon - h_2**	**Filtrere Innhold**
			00000000000
39 = 10 0111	011 = 3	101 = 5	00010100000
214 = 1101 0110	1110 = 14 => 3	1001 = 9	00010100010
353 = 01 0110 0001	11001=25=> 3	00100 = 4	00011100010

Hint: bruk $h(x)= y \bmod 11$, der y er hentet henholdsvis fra oddetalls-bits fra x eller partalls-bits fra x.

f) Anta at vi vil analysere de 11 siste meldingene som er kommet inn. Generelt på twitter, vil mange av meldingene bli sendt på nytt av samme bruker for å markere sitt synspunkt. Andre brukere vil "re-tweete" for å få flere til å få med seg meldingene. Forklar hvordan vi kan bruke bloom-filtre for å filtrere bort slike meldinger. Gjør de antakelsene du finner nødvendig.

Her kan vi sende hver av de 11 meldingene gjennom et visst antall hash-funksjoner som alle gir et integer tall. Vi starter med et bloom-filter som består av kun 0er. Når vi sender den første meldingen

gjennom første hashfunksjon, vil vi få et tall. Vi bruker det tallet og den biten som ligger i den posisjonen i bloom-filteret til 1 (dersom resultatet fra hash-funksjonen er 3, teller vi fra 0-3 fra venstre side i bloom-filteret, og setter den aktuelle biten til 1). Dette gjør vi for alle hashfunksjonene. Når vi har gjort dette for den første meldingen, går vi videre og gjør det samme for neste melding. Dersom vi får en melding hvor resultatet fra alle hash-funksjonene er sett før, altså at bitene i bloom-filteren alle er 1, så kan vi anta at meldingen er sett før. Dette kan likevel gi falske positiver, ved at en kombinasjon av andre meldinger har ført til at bloom-filteret består av 1ere på de aktuelle posisjonene. Vi kan derfor risikere at vi registrerer en melding som sett før selv om den egentlig er ny.

Det eneste vi kan finne ut helt sikkert med et bloom-filter er om en melding ikke er sett før. Dette gjelder dersom resultatet fra en av hashfunksjonene for en melding gir et tall, og man kan se at biten på den posisjonen i bloom-filteret er 0. Da kan vi si helt sikkert at meldingen ikke er sett før.

g) Anta nå at når de 11 meldingene har kommet inn har vi fått en strøm av data som ser ut som dette: 10100101010. Kan vi ha sett meldingen som kan representeres ved y = 1111011 før? Begrunn svaret ditt.

Oddetall: 1101 = 13 % 11= 2

Partall: 111 = 7

Denne meldingen kan ha vært sett før, siden bitene i posisjon 2 og 7 fra venstre side i filteret er 1.

Oppgave 6 – Anbefalingssystem (recommender systems) – 20 % (6% på a.i, 4% på a.ii, 10% på b)

Du er nyansatt i et nytt firma som vil spesialisere seg på strømming av film. En av oppgavene dine er å utvikle gode anbefalingsalgoritmer og metoder.

a) En del av metoden du foreslår går ut på å gi brukeren mulighet til å "rate" filmene for så bruke dette til å finne ut hvilke filmer systemet deres skal anbefale senere. Anta at brukerne deres har "rated" følgende 10 filmer med 3 eller flere stjerner:

```
Jurasic Park (Fantasi/SciFi), Harry Potter (Fantasi/Adventure),
ET (SciFi), Lord of the Rings (Fantasi/Adventure), Alien (SciFi),
Terminator (SciFi), 101 Dalmatians (Adventure/Family), Titanic
(Romantic), Sleepless in Seattle (Romantic) og Mr. Bean (Comedy).
```

i. Forklar hvordan du vil gå fram for å anbefale neste film til denne brukeren. Gjør de antakelsene du finner nødvendig.

Dersom metoden med rating har blitt implementert for alle brukere, ville jeg ha brukt item-item

anbefaling. Det vil si at jeg ser på hvordan de ulike filmene er ratet av de forskjellige brukerne, og bruker likhetsestimering for å finne en estimert rating for neste film.

ii. Ville du brukt innholdsbasert ("content-based") anbefalingsmetode eller "collaborative filtering"? Begrunn svaret ditt.

Her ville jeg brukt collaborative filtering, i og med at brukerne rater filmene og slik den nødvendige informasjonen er tilgjengelig. Collaborative filtering er generelt bedre enn content-based.

b) Anta følgende brukerratingstabell.

movies \ users	1	2	3	4	5	6	7	8
1	1		3			5		
2			5	4			4	
3	2	4		1	2		3	
4		2	4		5			4
5			4	3	4	2		
6	1		3		3		A	2

Bruk "***item-item collaborative filtering***"-metoden til å foreslå bruker nr. 7 sin rating av film nr. 6. Dvs. hva blir ratingverdien A? Du må vise mellomregningen.

Til denne oppgaven vil du trenge følgende formler:

Pearson Correlation similarity - likhet mellom vektor x, og vektor y:

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

der r_{xs} er bruker s sin rating på film x og $\overline{r_x}$ (overline) er gjennomsnitt av alle rating-ene på film x. **Vektet gjennomsnitt (weighted average)** for en brukers ratinger:

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

r_{ix} er her bruker x sin rating på film i, mens s_{ij} er likhet (similarity) mellom ratingene til film i og j

sim(6,1) = 0.53

sim(6,2) = 0.37

sim(6,3) = 0.05

sim(6,4) = 0.29

sim(6,5) = 0.41

sim(6,6) = 1

Vektet snitt = (4*0.37 + 3*0.05)/(0.37+0.05) = 3.88