LØSNINGSSKISSE TIL EKSAMENSOPPGAVE I FAG TDT4305 MAI 2021

NB! Dette er ikke fullstendige løsninger på oppgavene, kun skisse med viktige elementer, hovedsakelig laget for at vi skal ha oversikt over arbeidsmengden på eksamen, og som huskeliste under sensurering. Det er også viktig å være klar over at det også kan være andre svar enn de som er gitt i skissen som kan bli vurdert som korrekt om man har god grunngivning.

Oppgave 1 – Big Data-rammeverk

a) Diskuter i hvilken grad rammeverkene MapReduce, Spark, og Storm er egnet til sanntidsprosessering av innkommende data.

Denne oppgaven er relativt «open-ended», men viktig å også vise forståelse for *hvorfor* systemene er egnet eller ikke. Viktig er å vise forståelse for at MapReduce er batch-orientert og lite egnet. Spark kan brukes, men er ikke egnet der det er krav om lav responstid, Storm er i utgangspunktet strøm-basert og responstid er applikasjonsavhengig (men i utgangspunktet lav).

b) Hva kjennetegner de av transformasjonene («transformations») i Spark som har høy kostnad, og hva er årsaken/årsakene til høy kostnad?

De fører til «wide dependencies» som medfører shuffling og dermed ofte stor kommunikasjonskostnad.

c) Anta at du er ansvarlig for en Hadoop-klynge, bestående av 30 maskiner. Brukerne klager over at applikasjonene deres har lengre responstid enn før. Diskuter hva som kan være potensielle årsaker til dette, og hvor høy grad av sannsynlighet hver årsak har (høg/lav).

Relativt open-ended, men eksempel på typiske årsaker (med grad av sannsynlighet):

- Navnenode overbelastet (lav, evt. mer om man antar brukere som har generert mange små filer).
- Mye dataaksess som medføre mye kommunikasjon og gjør nettverk til flaskehals (høy).
- CPU-krevende applikasjoner som gjør prosessering til flaskehals (høy).
- Flere brukere enn før (høy).
- En eller flere noder er nede (lav til middels, avhengig av tidsperspektiv).
- Nettverksproblemer (lav).

Oppgave 2 – Shingles, minhashing, og LSH

Denne oppgaven kan løses på flere måter, avhengig av metode og tolking av tabellen for hash-funksjoner. Noen har brukt 1 som start på minhashing, slik at signaturene blir som nedenfor, men med +1 på alle verdier.

Metode 1: Ved å anvende den effektive/implementasjonsnære metoden (kap. 3.3.5):

Row	S0	S1	S2	S3	S4	h1(x)	h2(x)	h3(x)	h4(x)
0	1	0	1	1	1	1	3	0	2
1	1	1	0	0	0	3	1	2	0
2	0	0	0	1	1	2	0	3	1
3	1	1	0	0	0	0	2	1	3

	S0	S1	S2	S3	S4
h1	∞	∞	∞	∞	∞
h2	∞	∞	∞	∞	∞
h3	∞	∞	∞	∞	∞
h4	∞	∞	∞	∞	∞

R0	S0	S1	S2	S3	S4
h1	1	∞	1	1	1
h2	3	∞	3	3	3
h3	0	∞	0	0	0
h4	2	∞	2	2	2

R1	S0	S1	S2	S3	S4
h1	1	3	1	1	1
h2	1	1	3	3	3
h3	0	2	0	0	0
h4	0	0	2	2	2

R2	S0	S1	S2	S3	S4
h1	1	3	1	1	1
h2	1	1	3	0	0
h3	0	2	0	0	0
h4	0	0	2	1	1

R3	S0	S1	S2	S3	S4
h1	0	0	1	1	1
h2	1	1	3	0	0
h3	0	1	0	0	0
h4	0	0	2	1	1

	S0	S1	S2	S3	S4
Band 1	0	0	1	1	1
	1	1	3	0	0
Band 2	0	1	0	0	0
	0	0	2	1	1

I band 1 er (S0,S1) og (S3,S4) kandidater for match, i band 2 er (S3,S4) kandidater.
Regner ut Jaccard for (S0,S1) og (S3,S4), som gir hhv. 2/3 og 1. Dvs. vi har (S3,S4) som svar.
Legg her merke til at det er likhet mellom settene det spørres om, og ikke mellom signaturene.

Metode 2: Fremgangsmåten her er basert på minhashing (tilsvarende slide 33 fra forfatterne av læreboken), der man tenker seg at man går gjennom elementene i et sett iht. til hash-funksjon.
Eksempel:
For H1 og S0, rekkefølgen er rad 3, 0, 2, 1, første «1» treffes i første rad, og minhash-verdi er da 0.
For H2 og S0 er rekkefølgen rad 2, 1, 3, 0, først «1» treffes i rad 1, dvs. minhash-verdi er 1.
For H2 og S2 er rekkefølgen rad 2, 1, 3, 0, første «1» treffes i rad 0, dvs. minhash-verdi er 3.
For H4 og S2 er rekkefølgen rad 1, 2, 0, 3, første «1» treffes i rad 0, dvs. minhash-verdi er 2.
For H4 og S4 er rekkefølgen rad 1, 2, 0, 3, første «1» treffes i rad 2, dvs. minhash-verdi er 1.
Signaturene vil bli de samme som for metode 1, og dermed også resultat fra (b) det samme.

Metode 3: Her er tolkningen at hash-funksjonen gir en «re-ordering» av kolonnene, mens man teller fra første rad iht. til ny rekkefølge. Eksempel:
H1 og S2 er nå rad 0 den opprinnelige rad 1 (0), rad 1 er opprinnelig rad 3 (0), rad 2 er opprinnelig rad 2, rad 3 er opprinnelig rad 0 (1). Dvs. minhash-verdi er 3. Se under for resten av utregningene og signaturene med denne tolkningen:

x	h1(x)	S0	S1	S2	S3	S4
0	1	1	1	0	0	0
1	3	1	1	0	0	0
2	2	0	0	0	1	1
3	0	1	0	1	1	1
Min:		0	0	3	2	2

x	h2(x)	S0	S1	S2	S3	S4
0	3	1	1	0	0	0
1	1	1	1	0	0	0
2	0	1	0	1	1	1
3	2	0	0	0	1	1
Min:		0	0	2	2	2

x	h3(x)	S0	S1	S2	S3	S4
0	0	1	0	1	1	1
1	2	0	0	0	1	1
2	3	1	1	0	0	0
3	1	1	1	0	0	0
Min:		0	2	0	0	0

x	h4(x)	S0	S1	S2	S3	S4
0	2	0	0	0	1	1
1	0	1	0	1	1	1
2	1	1	1	0	0	0

3	3	1	1	0	0	0
Min:		1	2	1	0	0

Signaturer:

S0	S1	S2	S3	S4
0	0	3	2	2
0	0	2	2	2
0	2	0	0	0
1	2	1	0	0

LF:

	S0	S1	S2	S3	S4
Band 1	0	0	3	2	2
	0	0	2	2	2
Band 2	0	2	0	0	0
	1	2	1	0	0

I band 1 er (S0,S1) og (S3,S4) kandidater for match, i band 2 er (S0,S2) og (S3,S4) kandidater. Regner ut Jaccard for (S0,S1), (S0,S2) og (S3,S4), som gir hhv. 2/3, 1/3 og 1. Dvs. vi har (S3,S4) som svar.

NB!

- Legg her merke til at det er likhet mellom settene det spørres om, og ikke mellom signaturene (målet med hele prosessen er å finne sett med likhet over en viss terskelverdi, i dette tilfellet full match som er 1.0).
- Også viktig å vise at man forstår hvordan bruke båndene (dette er hovedpoenget med denne deloppgaven).

Oppgave 3 – Adwords

a) Hvorfor kan «competitive ratio» ofte forventes å være mindre enn 1 for on-line-algoritmer?

Strøm med spørringer, men vet ikke hva som kommer senere og må derfor ta en «endelig» beslutning umidddelbart for hver spørring (greedy).
NB! Vi er altså ikke bare ute etter, f.eks., definisjonen av comptitive ratio.

b) Gitt følgende tabeller med 1) annonsører og deres bud på spørringer, og 2) annonsører og deres budsjett:

Annonsør	Spørring	Bud
a1	q1	1
a2	q2	0.5
a2	q3	0.5
a3	q2	1

a4	q1	0.75
a4	q2	0.5
a4	q4	0.5

Annonsør	Budsjett
a1	1
a2	3
a3	1
a4	2

Anta følgende spørringer, i gitt rekkefølge: q1, q2, q4, q3, q2, q2, q3, q2, q2

Finn annonsør-spørring-par ved å bruke «Balance»-algoritmen. Forklar hvordan du bruker algoritmen for å komme frem til resultatene, og hva som blir akkumulert inntekt.

Spørring	Kandidater & bud	Gjenværende budsjett	Akkum. inntekt
q1	(a1, 1), **(a4, 0.75)**	B4=1.25	0.75
q2	**(a2, 0.5),** (a3, 1), (a4,0.5)	B2=2.5	1.25
q4	**(a4, 0.5)**	B4=0.75	1.75
q3	**(a2, 0.5)**	B2=2	2.25
q2	**(a2, 0.5),** (a3, 1), (a4, 0.5)	B2=1.5	2.75
q2	**(a2, 0.5),** (a3, 1), (a4, 0.5)	B2=1	3.25
q3	**(a2, 0.5)**	B2=0.5	3.75
q2	(a2, 0.5), **(a3, 1),** (a4, 0.5)	B3=0	4.75
q2	(a2, 0.5), **(a4, 0.5)**	B4=0.25	**5.25**

Noen få har brukt «Generalized Balance», dette er godtatt.

Oppgave 4 – Systemer for datastrøm

1. Drøft hvorfor et system som AsterixDB Feeds, Spark og Storm er nødvendige for å muliggjøre håndteringen av datastrøm. Forklar hvilke fordeler og ulemper hver av disse systemene har.
   Svar:
   Dedikerte systemer for å håndtere datastrøm er nødvendig for å klare å hanskes med karakteristikkene av strøm av big data, som skalering, hastighet («High velocity»), og at data kommer uten stopp og uten grenser, samt har forskjellige format og struktur.

   Fordeler (+) og ulemper (-) med systemene:

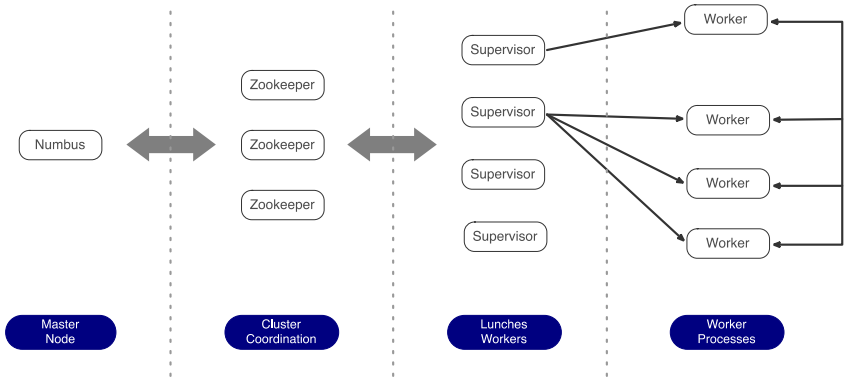
   AsterixDB:
   + Et komplett system med innebygd effektiv lagringshåndtering av bigdata som også kan utvides med eksisterende lagringssystemer
   + Sofistikerte «recovery»-mekanismer som kan håndtere både «soft og hard failures»
   + Tillater å ta inn høyhastighetsdata med effektiv håndtering

+ Lar man håndtere data både i batch og ren strøm (sanntidsbehandling/analyse) form
+ Feed kan programmeres til å håndtere én input-strøm og produsere flere strøm av resultat-outputs («fetch once, compute many» prinsippet).
- Komplisert – kan være for mange moving parts, noe som gjør det vanskelig å vedlikeholde effektivt
- Proprietære spørringsspråk
- Ikke like utbredt som Storm og Spark

Spark:
+ Veldig enkel å ta i bruk
+ Kan kobles til standard og eksisterende systemer som kan gi mye fleksibilitet
+ Micro-batching kan gjør de veldig naturlig med «windowing» (aka. sliding windows)
- Bruker tupler som kan være litt vanskelig sammen Java
- Micro-batching gjøre det ikke rettfrem med håndtering og analyse i sanntid
- Uten innebygd naturlig datalagring og recovery-håndtering
- Fortsatt «single point of failure» i strømmen

Storm:
+ Veldig enkel å ta i bruk
+ Kan kobles til standard og eksisterende systemer som kan gi mye fleksibilitet
+ Sanntidsstrømhåndtering
+ Innbygd user-defined function-mulighet som gjøre det mulig å implementer funksjoner selv.
- Uten innebygd naturlig datalagring og recovery-håndtering
- Sanntidsfokus gjøre det litt vanskelig å implementer «windowing»

2. Tegn og forklar *systemarkitekturen* til Storm (Tips: ikke topologi).
   Svar:



3. Forklar hva som menes med «Delivery Semantics (Message Guarantees)» for datastrøm. Bruk eksempler til å støtte forklaringen din.
   Svar:
   Skal forklare følgende (Akseptable eksempler forventes)

   At most once: meldinger kan være bort under sending men aldri sendes på nytt. Eks. sending av data fra værstasjoner som mottas som grunnlag for værmelding. Det er ikke kritiske at all data når frem så lenge man har en sample av data.
   At least once: meldinger blir aldri bort og kan sendes på nytt.

Eks. sending av data som brukes til vedlikehold av elektriske systemer. Om man har alt datagrunnlag kan man ikke predikere når det er nødvendig med vedlikehold. Det her viktig at man har alt av data tilgjengelig med det er ikke livsviktig at de blir levert i sanntid.
Exactly once: meldinger bli aldri bort og blir aldri sendt på nytt, dvs. den mest ideelle medlingsleveringen. Eks. data fra sensorer på kritiske systemer som må håndteres i sanntid (som feks. data fra høydesensorer på fly) og er livskritisk hvis ikke bli levert.

4. I sammenheng feiltoleranse av et system for datastrøm skiller vi mellom «soft failure» og «hard failure». Forklar forskjellene mellom disse.
   Svar:
   Soft failure handler om systemfeil som kan skyldes programvarefeil eller datafeil. I en strøm av data kan dette handle om feil i levering av data som skyldes bugs i en kode som er skrevet av en bruker.
   Hard failure handler om systemfeil som kan skyldes at en systemnode er nede. Dette kan syldes hardwarefeil i en eller flere node.

Oppgave 5 – Håndtering av datastrøm

For å overvåke uønskede aktiviteter bruker sosialemediaplatformer som Instagram og Twitter algoritmer som analyserer bruksmønstre som feks. hvor ofte brukerne bl.a. trykker på likes eller kommenterer på en post/melding/bilde i løpet av en gitt periode til å avgjøre om de er reelle brukere eller «bots», for så å straffe de med utestengelse. For eksempel utestenger Instagram ofte brukerne sine midlertidig hvis de kommenterer for mange/et spesifikt antall poster i løpet av en time, flere ganger i løpet av et døgn. Dette for å rense plattformen for spammere ol.

Anta at du skal hjelpe Instagram med oppgaven over, dvs. å finne ut hvilke brukere som kan være bots. *Gjør ellers de antakelsene som du mener er nødvendige* og svar på følgende spørsmål:

I svarene er det viktig at studenten viser forståelse for metodene.

1. Instagram-oppgaven kan løses ved hjelp av både stående spørring (standing queries) og ad-hoc spørring. Forklar hvordan.
   Svar:
   Stående spørring: Man kan lage en algoritme som skal *kontinuerlig* spørre om det er kommet en ny likes, legge dette sammen med ant. likes så langt og returnere resultat når antall overskrider en gitt verdi. Spørringen må det vite når første like var registrert og sammenlikner dette med nåværende tidsstempel.

   Ad-hoc: Man kan lage en algoritme som skal spørre *en gang* hver time og sender ut resultat dersom summen av likes overskrider en gitt verdi.

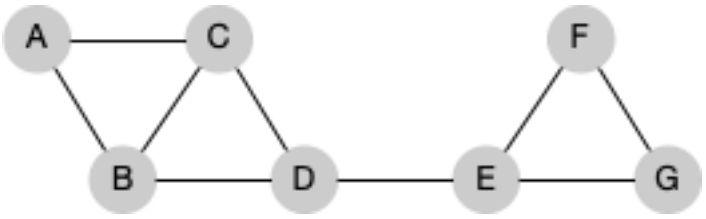
2. Skisser en løsning for Instagram basert på glidene vinduer (sliding windows).
   Svar:
   Her må man anta at man har en strøm av bilder i brukerens feed. Man kan da implementere dette som bit counting med glidende vindu. For vær gang brukeren trykker på like registreres

dette som «1», «0» ellers. Anta videre at minnestørrelse ikke er noe stort problem. Da kan man bruke tidsvindu på en time som tidsbasert vindustørrelse. Her vil problemet da vær løst ved å summere antall enere innenfor vinduet.

3. Kan Flajolet-Martin-algoritmen brukes her til å estimere antall «likes» til en bruker? Forklar.
   Svar:
   Ja, selv om kanskje foregående løsning ville være mindre komplisert. Her kan man løse oppgaven ved å estimere hvor mange *forskjellige* bilder (i en strøm av mange mange bilder) en bruker har trykket «like» på i løpet av en time, og eller i løpet av en dag.

Oppgave 6 – Sosiale grafer og Anbefalingssystemer

1. Gitt følgende forenklet sosiale graf. Bruk Girvan-Newman-metoden til å finne klynger/«communities» i grafen.



   Svar: Denne oppgaven kan liten modifisering av den som er gjennomgått på forelesning hvor man skal gjennomføre følgende:
   Girvan-Newman Algorithm:
   - Gjenta følgende til det er ingen kanter igjen i grafen:
     - Beregn betweeness for kantene
     - Fjern kantene som har høyeste betweeness
     - Komponenter som har koblinger er «communities». Her vil man da få hierarkiske «communities»

2. Man velger ofte item-item collaborative filtering fremfor user-user collaborative filtering fordi den er mer effektiv. Er du enig? Hvorfor/hvorfor ikke?
   Svar: item-item collaborative filtering velges ofte fremfor user-user collaborative filtering. Sammenlikning av produkt og hvordan de er blitt likt av brukere er ofte enklere og dermed mer effektive. Dette vil man kan modellere og lag produktprofiler basert på målbare og deterministiske data som likes, etc. Brukere har derimot forskjellige smak som ikke nødvendigvis er konsekvent og derfor er mer komplisert enn produkt.

3. Det finnes tre likhetsmål (similarity measures) som man kan bruke for å sammenlikne brukere eller produkt i forbindelse med anbefaling av et produkt til en bruker. Drøft hvilke likhetsmål disse er. Forklar kort hvilket av disse likhetsmålene er minst egnet til bruk til anbefaling. Bruk et konkret eksempel til å støtte forklaringen din.
   Svar:
   - Jaccard likhetsmål som måler likhet mellom to brukere basert på *andelen av* alle produkt de har ratet.

- Cosinus likhetsmål som ser på sett av produkt som brukere har ratet og verdiene av ratingene som vektorer. For å måle likhet mellom to brukere ser man da cosinus av vinklene på ratingsvektorene for disse to brukerne
- Normalized cosinus eller Pearson Correlation ser i likhet med Cosinus på sett av produkt som brukere har ratet og verdiene av ratingene som vektorer. I stedet for å bare bruke verdiene av ratings trekkes snittet av alle ratings fra ratingverdiene før man beregner cosinusverdiene. Brukes pearson correlation ser man kun på verdiene der begge brukere har ratet de samme produktene.