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:

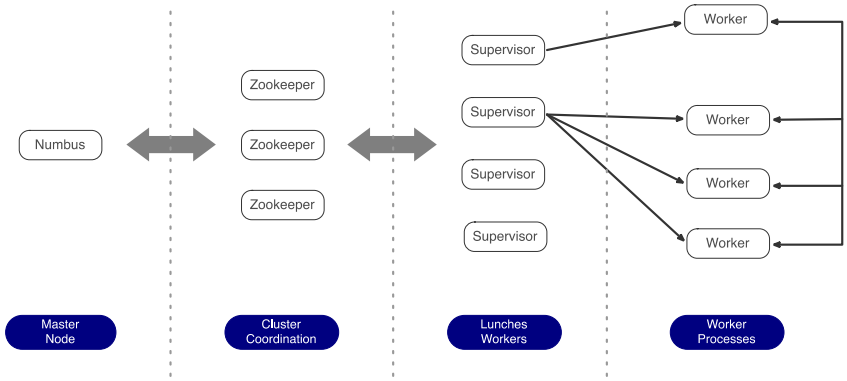
   AsterixDB:
   + Et komplett system med innebygd effektiv lagringshåndtering av bigdata som også kan utvides med eksisterende lagringssystemer
   + Sofistikerte «recovery»-mekanismer som kan håndtere både «soft og hard failures»
   + Tillater å ta inn høyhastighetsdata med effektiv håndtering

+ 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.

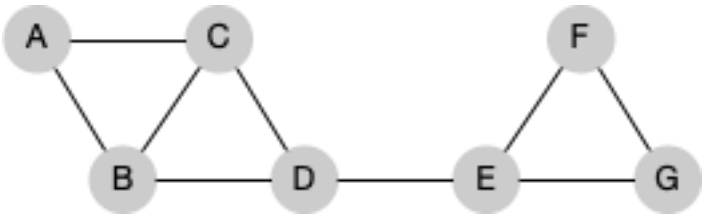
2. Skisser en løsning for Instagram basert på glidene vinduer (sliding windows).
   Svar:
   Her må man anta at man har en strøm av bilder i brukerens feed. Man kan da implementere dette som bit counting med glidende vindu. For vær gang brukeren trykker på like registreres

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.

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:

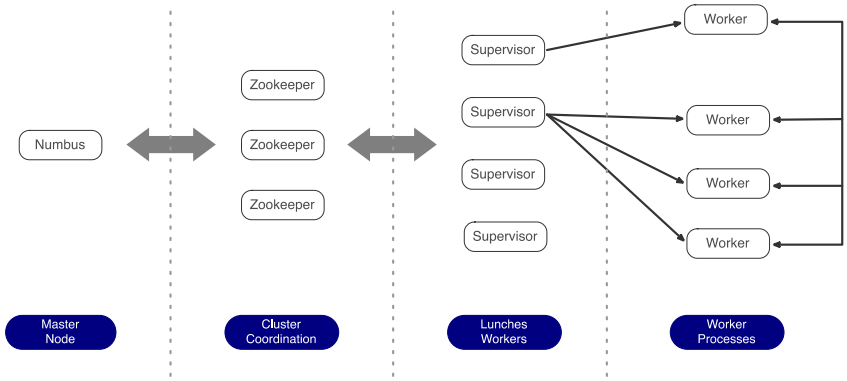
   AsterixDB:
   + Et komplett system med innebygd effektiv lagringshåndtering av bigdata som også kan utvides med eksisterende lagringssystemer
   + Sofistikerte «recovery»-mekanismer som kan håndtere både «soft og hard failures»
   + Tillater å ta inn høyhastighetsdata med effektiv håndtering

+ 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.

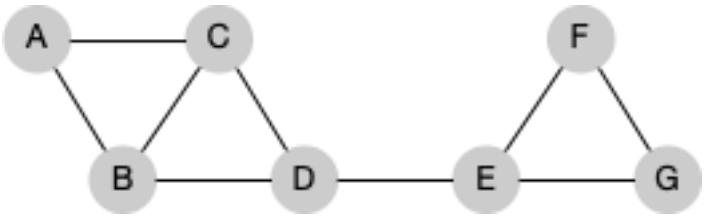
2. Skisser en løsning for Instagram basert på glidene vinduer (sliding windows).
   Svar:
   Her må man anta at man har en strøm av bilder i brukerens feed. Man kan da implementere dette som bit counting med glidende vindu. For vær gang brukeren trykker på like registreres

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.

Cover Page

Department of Computer Science

Ordinary Exam in TDT4305 Big Data Architecture

Contact during the exam: Heri Ramampiaro
Phone: 99027656

Date for the exam: 27th of May 2019
Time: 09:00 – 13:00

Permitted aids: D – No printed or hand-written support material is allowed. A specific basic calculator is allowed.

Read all questions carefully. Make any necessary assumptions wherever these are necessary.

Problem 1 – Big Data (10%)

1. Along with the term Big Data, we often refer to the three or four V's, and "Velocity" is one of such V's. What does Velocity stand for and why is this important to consider in relation to handling Big Data? Use example to support your explanation.

Answer:

- Velocity stands for speed, i.e., the speed of data creation from a given source such as social media messages and sensors.
- Velocity is important to consider since the data are also normally created in a high speed, the come as open ended and in a large volume, putting high demands on processing efficiency and scalability, in addition to hardware requirements. For example, when computing the average of all streaming tweets or Instagram messages containing a specific topic, we need to consider the fact that the comes in all the time in a speed that might be unbound. This makes it necessary to put some boundary, e.g., in terms of a window size based on time.

2. Briefly name and explain at least two examples of systems used to handle Big Data.

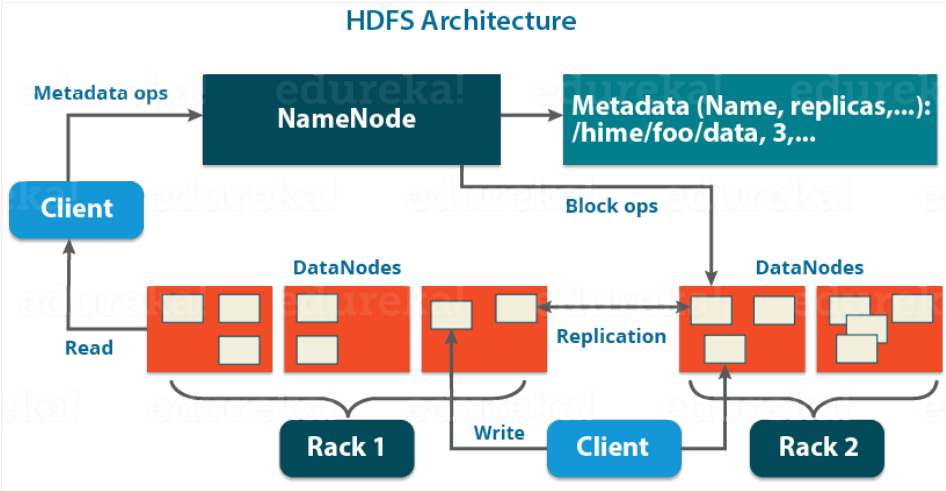
Answer:

A system to handle big data could be a framework, platform or a software, i.e., Hadoop, Spark, AsterixDB, etc..

Problem 2 – Hadoop (10%)

1. What is Hadoop and explain briefly how the Hadoop Distributed File System (HDFS) is built up.

Answer:
Hadoop Distributed File System (HDFS) is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. HDFS follows the master–slave data architecture. Each cluster comprises a single Namenode that acts as the master server in order to manage the file system namespace and provide the right access to clients.



2. Explain what the roles of "NameNode" are in HDFS.

Answer:
NameNode is, as can be inferred above, the master node in the HDFS architecture. It maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients.

3. How is error tolerance taken care of with HDFS?

Answer:
HDFS provides a reliable way to store huge data in a distributed environment as data blocks. To ensure fault tolerance, these blocks are replicated across different data nodes.

Problem 3 – MapReduce og Spark (20%)

1. Explain what the ideas behind MapReduce are (5%).

Answer:

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map function* that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce function* that merges all intermediate values associated with the same intermediate key.

Programs written in this functional style are *automatically parallelized and executed on a large cluster of commodity machines*. The run-time system (e.g. Hadoop) takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication, thus allowing programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

2. In the social media platform, LinkedIn, one often wants to find lists of connections, often called "connections". Such a connection is bi-directional, i.e., if you have a connection with a person, this person will also have a connection with you. Now, suppose, for example, we have the following lists of people and their connection:

   Adrian:Bjørn, Camilla, David
   Bjørn: Adrian, Camilla, David, Ester
   Camilla: Adrian, Bjørn, David, Ester
   David: Adrian, Bjørn, Camilla, Ester
   Ester: Bjørn, Camilla, David.

   Quite often when you try to get in touch with a person on LinkedIn, you want to find out which other persons this person has a common connection with you. This necessitate the ability to look up such information efficiently. Show how MapReduce can be used for this purpose. (10%).

Answer:

Note: *Students have to show they understand the concept. To get a full score, they have to show how map-reduce procedure is concretely done using the LinkedIn connection task. This means that only explaining a solution sketch will not give any full score.*

LinkedIn has a list of *connections*, and as noted connections are a bi-directional – meaning that if I'm your connection, you're mine. When you visit someone's profile, you see a list of connections that you have in common. Assuming this list doesn't change frequently, it would be wasteful to recalculate it every time you visited the profile. This is why MapReduce would be suitable for the calculation of everyone's common connections so that later on it's just a quick lookup, and in our case, this will be a two-names pair, i.e., (name1, name2).

We assume that the connections are stored as `Person: [List of Connections]`, and that they are as follows:

```
Adrian: Bjørn, Camilla, David
Bjørn: Adrian, Camilla, David, Ester
Camilla: Adrian, Bjørn, David, Ester
David: Adrian, Bjørn, Camilla, Ester
Ester: Bjørn, Camilla, David.
```

Mapper:
Each line will be an argument to a mapper. For every connection in the list of connections, the mapper will output a key-value pair. The key will be a connection along with the person (two-names pair). The value will be the list of connections. The key will

be sorted so that the connections are in order, causing all pairs of connections to go to the same reducer. After all the mappers are done running, you'll have a list like this:

For **`map(Adrian: Bjørn Camilla David)`**:
```
(Adrian Bjørn): Bjørn Camilla David
(Adrian Camilla): Bjørn Camilla David
(Adrian David): Bjørn Camilla David
```

For **`map(Bjørn: Adrian Camilla David Ester)`**: *(Note that Adrian comes before Bjørn in the key)*
```
(Adrian Bjørn): Adrian Camilla David Ester
(Bjørn Camilla): Adrian Camilla David Ester
(Bjørn David): Adrian Camilla David Ester
(Bjørn Ester): Adrian Camilla David Ester
```

For **`map(Camilla: Adrian Bjørn David Ester)`**:
```
(Adrian Camilla): Adrian Bjørn David Ester
(Bjørn Camilla): Adrian Bjørn David Ester
(Camilla David): Adrian Bjørn David Ester
(Camilla Ester): Adrian Bjørn David Ester
```

For **`map(David: Adrian Bjørn Camilla Ester)`**:
```
(Adrian David): Adrian Bjørn Camilla Ester
(Bjørn David): Adrian Bjørn Camilla Ester
(Camilla David): Adrian Bjørn Camilla Ester
(David Ester): Adrian Bjørn Camilla Ester
```

And finally for **`map(Ester: Bjørn Camilla David)`**:
```
(Bjørn Ester): Bjørn Camilla David
(Camilla Ester): Bjørn Camilla David
(David Ester): Bjørn Camilla David
```

Before we send these key-value pairs to the reducers, we group them by their keys and get:
```
(Adrian Bjørn): (Adrian Camilla David Ester) (Bjørn Camilla David)
(Adrian Camilla): (Adrian Bjørn David Ester) (Bjørn Camilla David)
(Adrian David): (Adrian Bjørn Camilla Ester) (Bjørn Camilla David)
(Bjørn Camilla): (Adrian Bjørn David Ester) (Adrian Camilla David
Ester)
(Bjørn David): (Adrian Bjørn Camilla Ester) (Adrian Camilla David
Ester)
(Bjørn Ester): (Adrian Camilla David Ester) (Bjørn Camilla David)
(Camilla David): (Adrian Bjørn Camilla Ester) (Adrian Bjørn David
Ester)
(Camilla Ester): (Adrian Bjørn David Ester) (Bjørn Camilla David)
(David Ester): (Adrian Bjørn Camilla Ester) (Bjørn Camilla David)
```

Reducer:
Each line will be passed as an argument to a reducer. The reduce function will simply intersect the lists of values and output the same key with the result of the intersection. For example, `reduce((Adrian Bjørn): (Adrian Camilla David Ester) (Bjørn Camilla David))` will output `(Adrian Bjørn) : (Camilla David)` and means that connections Adrian and B have C and D as common connections.

The result after reduction is:
```
(Adrian Bjørn): (Camilla David)
```

```
(Adrian Camilla): (Bjørn David)
(Adrian David): (Bjørn Camilla)
(Bjørn Camilla): (A David Ester)
(Bjørn David): (A Camilla Ester)
(Bjørn Ester): (Camilla David)
(Camilla David): (A Bjørn Ester)
(Camilla Ester): (Bjørn David)
(David Ester): (Bjørn Camilla)
```

Now when David visits Bjørn's profile, we can quickly look up (Bjørn David) and see that they have three connections in common, (Adrian Camilla Ester).

3. Explain briefly how you would solve the same problem using Spark. Make any assumptions that you think are necessary. (5%).

Answer/Solution sketch:
Here the student has to show how he/she creates an RDD(Resilient Distributed Dataset) of input strings using the information above. Thereafter they have to explain how the input is used for the map and reduce functions. An answer here could look like a pseudo code.

Problem 4 Systems for data stream (10%)

1. Explain the main differences between Spark and Storm. Discuss especially when Storm is more suitable than Spark and vice versa.

Answer/Solution sketch:
Explanation can be based on the following table:

	Storm	**Spark**
Processing Model	Event-Streaming	Micro-Batching / Batch (Spark Core)
Delivery Guarantees	At most once / At least once	Exactly Once
Latency	Sub-second	Seconds
Language Options	Java, Clojure, Scala, Python, Ruby	Java, Scala, Python

Development	Use other tools for batch	Batching and streaming are very similar

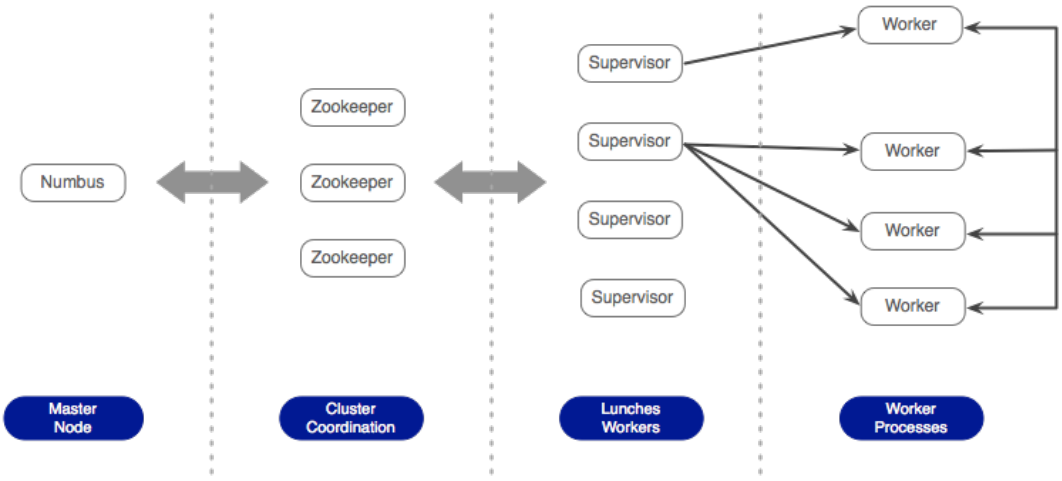
Based on this table, Spark is recommended when we need an iterative (mini) Batch Processing as in most Machine Learning tasks and when the latency requirements is seconds and not less. In contrast, Storm is recommended when we need a real-time processing, with a sub-second latency requirement.

2. Explain Storm's system architecture. Use the figure to support your explanation

Answer/Skecth:

Storm's system architecture consists of several components as shown in the figure below.

- *Nimbus* is a master node of Storm cluster, which is responsible for distributing data among all the worker nodes, assign tasks to worker nodes and monitoring failures.
- *ZooKeeper* is a service used by a cluster (group of nodes) to coordinate between themselves and maintaining shared data with robust synchronization techniques. Since, nimbus is stateless, it depends on ZooKeeper to monitor the working node status. In addition Zookeeper helps the supervisors to interact with nimbus.
- *Supervisor*s are the nodes that follow instructions given by the nimbus. A supervisor has multiple worker processes and it governs worker processes to complete the tasks assigned by the nimbus.
- *Worker* will execute tasks related to a specific topology. A worker process will not run a task by itself, instead it creates *executors* and asks them to perform a particular task.



3. Discuss how AsterixDB can be used to replace Spark and Storm.

Answer:

AsterixDB is a BDMS (Big Data Management System) with a rich feature set that sets it apart from other Big Data platforms. This set of feature includes handling streaming data

through so-called data feed and user-defined function. A data feed here is a flow of data from an external source into persistent (indexed) storage inside a Big Data Management System (BDMS). AsterixDB replaces Spark and Storm in providing the possibility not only to handle inputs of data streams, but also the ability to process the input data in both real-time and through batches, with high performance and scalable UDFs. In addition, AsterixDB provides a tool/system for persisting the input and processed data in a database using the AsterixDB data model.

Problem 5 Social network (15%)

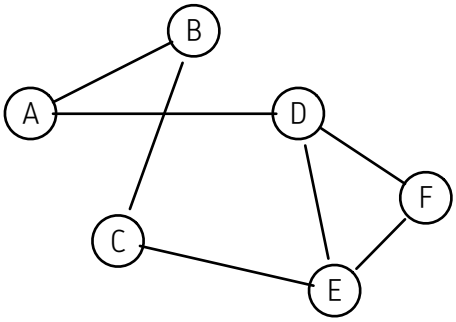


Figure 1 Social network graph

1. What are the main purposes of **community detection** in social networks? (2%)

Answer:
Community detection in social network is key to understanding the structure of complex networks, and ultimately extracting useful information from them. One of its main concern is to detect clusters in a graph making it useful for, e.g., to communality among people connected in the network. This in turns could be used for recommendation or other decision support purposes.

2. Construct an **adjacent matrix** based on Figure 1. (5%)

Answer:
An adjacent matrix A of a graph is an n x n matrix, with which n is the number of nodes in the graph, and $A = [a_{ij}]$, $a_{ij} = 1$ if there is an edge between a node i and j. Based on this the adjacent matrix for our graph in Fig. 1 would be given as in the following table:

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	0	0
C	0	1	0	0	1	0
D	1	0	0	0	1	1
E	0	0	1	1	0	1
F	0	0	0	1	1	0

3. What do we mean by **edge betweenness**? Use the Girvan-Newmann method to find the edge betweennesses in the network illustrated in the figure above (Figure 1). Make the assumptions that you think are necessary. (8%)

Answer/Solution sketch:
Edge betweenness is the number of shortest paths passing through the edges. It is used to identify possible communities in a graph. Using the Girvan-Newmann method, we can compute the edge betweennesses in a graph through the following steps:

- Repeat until no edges are left:
  - Calculate betweenness of edges
  - Remove edges with highest betweenness
- Connected components are communities
- This gives a hierarchical decomposition of the network

To get a full score an answer must contain a more detailed description of how to compute the betweenness of edges.

Problem 6 Data Stream 1 (10%)

1. Standing queries are a term often used in relation to querying data streams. Explain briefly what this is about and illustrate with example how this can be used. (5%)

Answer:
Standing queries are queries that are, in principle, asked about the stream at all times. Example: Report each new maximum value ever seen in stream S.

2. Explain how we can use "Bit Counting" to find out the fraction of Instagram users who mention the word "iPhone" in their Instagram messages. Make any assumptions you find necessary. (5%)

Answer:
Bit counting concerns solving the following problem: Given a stream of 0's and 1's, be prepared to answer queries of the form "how many 1's in the most recent k bits?" where $k \leq N$. Translated to our question, we get each time an Instagram message mentions/contains iPhone we set stream bit to "1" if this is the case, "0" otherwise. We will then eventually get a stream of 1s and 0s. To find the fraction Instagram users who mention the word "iPhone", we could then just count the number 1s in the total length of binary stream seen so far.

Problem 7 Data Stream 2 (10%)

1. Explain the principle behind bucket representation of data streams. Use illustration to support your explanation. (5%)

Answer:
A bucket representation of a stream is *a segment of a stream window*, represented by a record consisting of:

- The timestamp of its end [O(log N) bits].
- The number of 1's between its beginning and end, which also constitutes the size of the bucket.

2. Use the "bloom filter" principle to fill in the table below (5%).

Stream elements	Hash-function - *h1*	Hash- function - *h2*	Filter content
			00000000000
57 = 11 1001			
214 = 101 1110			

Hint: use *h(x)= y mod 11*, where *y* is computed from the odd and even bits of *x,* respectively.

Answer:
57=1**1** 1**0**0**1** => h1: 101=5 mod 11 = 5 ⇔ h1=5; h2: 110=6 mod 11 = 6 ⇔ h2=6
(1): 00000110000

214=**10**1 1**1**1**0** => h1: 1110=14 mod 11 = 3 ⇔ h1=3; h2: 011=3
(2) 00010110000

Problem 8 Recommender Systems (15%)

Musify is a start-up company that specializes in music streaming. Their business idea is to let users stream music freely, given that they give feedback on how they like the music they listen to and that Musify can use this for music recommendation and marketing.

1. Musify has the choice of using content-based recommendation and collaborative filtering. Which of these methods would you prefer? Explain. (5%)

Answer:
A key element here looking at the description is the fact that users give feedbacks on how they like the music they listen to. Based on this it is safe to assume that they could either give explicit feedback or implicit feedback. For simplicity we assume the former, and that this is given through rating the music from 1 to 5. With the availability of user feedback the most sensible solution is to use collaborative filtering (CF). The fact that there is no information about how the music are represented – i.e., which info of the music is available, makes the content-based recommendation a harder choice, though this would be preferred to a lowest degree of cold start issues.

2. Assume that Musify selects "collaborative filtering" and that the table below shows how five of their users (U1 - U5) have "rated" the music (M1 - M5) they have listened to from 0 to 10.

User Is/Music-Id	U1	U2	U3	U4	U5	Cosine(i, 1) (user-user)
M1	7	6	7	4	5	
M2	6	7	0	4	3	
M3	0	3	3	1	1	
M4	1	2	2	3	3	
M5	1	0	1	2	3	

We assume that we will use user-user collaborative filtering. (a) What do we mean by user-user collaborative filtering? (b) What another alternative method do we have? (c) Use cosine equality measure to fill in the table above (10%).

Answer:

(a) User-user collaborative filtering is collaborative filtering with focus on the similarity among users – i.e., their preferences. This means that we recommend a given item/product to a given user based on how similar other users have previously rated this item.

(b) Another alternative to user-user CF would be item-item CF. In contrast to the user-user alternative, item-item focuses on the similarity among the items. This means that we recommend a given item/product to a given user based on how other similar items have previously been rated as compared to this item.

(c)

$$\cos(1,2) = \frac{[7,6,0,1,1] \cdot [6,7,3,2,0]}{\sqrt{7^2+6^2+1^2+1^2}\sqrt{6^2+7^2+3^2+2^2}} = \mathbf{0.93}$$

Using similar computation, we get:
Cos(1, 3)=0,70
Cos(1, 4)=0,90
Cos(1, 5)=0,87

Du skal bruke følgende formel til beregningen din:

$$\boldsymbol{sim(x,y) = cos(r_x, r_y) = \frac{r_x \cdot r_y}{\|r_x\| \cdot \|r_y\|}}$$

LØSNINGSSKISSE TIL EKSAMENSOPPGAVE I FAG TDT4305 – MAI 2018

NB! Dette er ikkje fullstendige løysingar på oppgåvene, kun skisse med viktige element, hovudsakleg laga for at vi skal ha oversikt over arbeidsmengda på eksamen, og som huskeliste under sensurering. Det er også viktig å være klar over at det også kan vere andre svar ein dei som er gjeve i skissa som vert rekna som korrekt om ein har god grunngjeving.

Oppgåve 1 – Hadoop – 10 %

a) Replikering av blokker og bruk av sjekksum på kvar blokk for å detektere diskfeil.
b) Combiner utfører delvis samanslåing av par med same nøkkel fra mapper (oftast vha. same funksjon som Reduce) , slike at desse vil verte prosessert som gruppe av Reduce-task.
   Kan redusere kommunikasjonskostnad.
c) Reduserer kommunikasjonskostnad for skrive-klient som kun sender data til ei anna node i staden for til tre, og i tillegg der ein har fleire rack vil ein redusere kommunikasjon mellom rack (om to replika vert skrive til remote rack vil ein kun ha behov for ein inter-rack kommunikasjon).

Oppgåve 2 –Spark – 10 %

Kan også besvares med Python eller Java. "Utskrift" i oppgåva er gjeve for å vise innhald i resultat-RDD, ikkje noko krav om formatering i studentane sin kode.
Generelt: Ikke trekk for mindre "språk-typo" (hensikta med oppgåva er å vise at ein forstår Spark, men trekk om det er gjeve fleire svaralternativ der eitt eller fleire er feil (forsøk på "gardering").
OK om "add" er brukt i Python i staden for "_+_", ein variant av countByKey kan også brukast istaden for reduceByKey.
Trekk for dei som ikkje forstår at det er arrays og ikkje tuplar i start-RDD.
Gjeve: val s = sc.textFile("streamed.csv").map(_.split(","))

a) val data1 = s.map(a => (a(2),1)).reduceByKey(_ + _)
b) val data2 = s.map(a => (a(1),1)).reduceByKey(_ + _)
c) val data3 = s.map(a => (a(3))).distinct().count // OK om også a(2) er med slik at artist+song er "nøkkel"

Oppgåve 3 – NoSQL – 15 %

a) Key-value stores: Data-modell basert på aksess til *verdi* (typisk post, objekt, eller dokument, men kan også ha meir kompleks data-struktur) via *nøkkel.*
   Document stores: Lagrar data som dokument i format som, t.d., JSON. Aksesserast via dokument-identifikator. Indeksering.
   Extensible record stores/BigTable clones: Tabell partisjonert i kolonne-familiar, der kvar kolonne-familie lagra i eigen fil.
   Graph Databases: Data representert som graf, der relaterte noder kan finnast ved å traversere kantar vha. sti-uttrykk.
b) Jfr. 24.4.2 i Elmasri&Navathe. Sentrale element: *sharding* og *horisontal skalerbarheit vha. konsistent hashing* (som gjev lastbalansering og feiltoleranse). Forventa at "ringen" er teikna/forklart.

Oppgåve 4 – MinHashing – 10 %

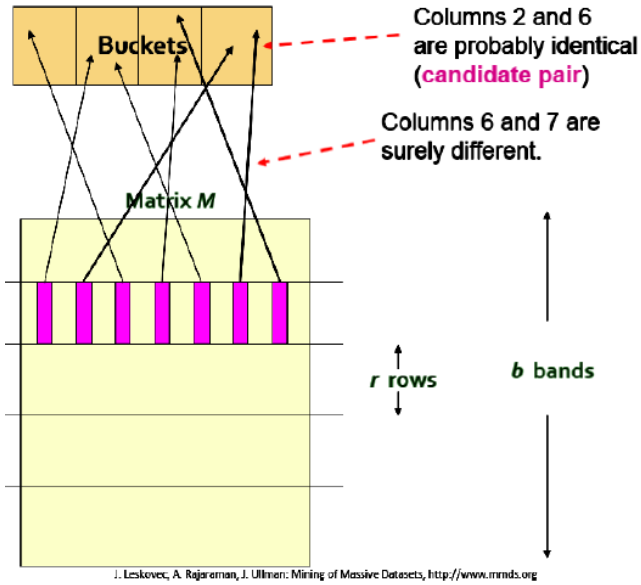
(Som det også vart informert om på eksamen så er tittelen litt misvisande, det er ikkje nødvendig å forklare shingling og MinHashing, vi er kun ute etter detaljane for LSH, og det er kun svar relatert til LSH som gjev poeng).

Jfr. 3.4 i MMD.

Hensikt: Effektivt finne dokument-par som er potensielle "nær-duplikat", alternativet er f.eks. å sjekke alle par av dokument (evt. signaturane deira) mot kvarandre.

Input: Signatur-matrise M (Docs x Signatures), Output: Liste med kandidat-par S1, S2 som må evaluerast med s(S1, S2)

Datastruktur: Matrisa M er delt inn i *b* "bands", kvar med *r* rader:



For kvart dokument, for kvar band: hash denne delen av signaturen inn i hash-bøtte (Separate bøtter for kvart band).

Gå gjennom alle bøtter for alle band: Konstruer kandidat-par for alle dokument i bøtte.

Oppgåve 5 – Adwords – 5 %

Jfr. kap. 8 i MMD:

Eit sett med bod for kvart søkeord, budsjett for kvar annonsør, eit sett med brukarar og deira søkeord.

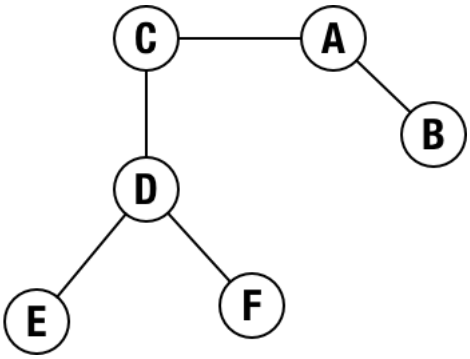
Adwords-problemet er å gje match mellom spørjing og søkeord på ein slik møte at inntekt vert maksimert.

Grådig algoritme: vel tilfeldig annonsør som har bod for søkeord i spørjing. Ulempa med denne er at ein annonsør med mange søkeord kan bruke opp budsjettet sitt og det kun er annonsørar med avgrensa (og ikkje søkte etter) søkeord att.

Endring: Vel annonsør som har høgst attverande budsjett.

Oppgave 6 (15%)

1. Beskriv og forklar kort hvilke steg du trenger å kjøre for å finne «community» i en graf. Bruk følgende figur til å støtte forklaringen din. (6%)



Svar:

In this task, it is expected that the students not only describe the algorithm but also is able to tell which method he/she uses to solve it. There are several methods for community detection. One possible method is Girwan-Newman algorithm. To get a full score the answer need to show in full that he/she understand the method.

1. Forklar hovedhensiktene med å analyser sosiale grafer generelt. (4%)

Svar:

Here the expected answer would be to show that the student understand what it means to analyse social graphs. On goal would be to identify community, e.g. to understand how users in a social network are connected. This could be useful for group recommendation, social interactions and preferences, etc. Another goal could be to simply partition graphs to reduce the amount of information to be processed (e.g. through graph clustering etc.).

Forklar hovedforskjellene mellom Storm og Spark. Forklar hvilke fordeler AsterixDB sitt Feed-system har i forhold til både Storm og Spark. (5%)

Svar:

Storm vs. Spark:

	Storm	**Spark**
Processing Model	Event-Streaming	Micro-Batching / Batch (Spark Core)
Delivery Guarantees	At most once / At least once	Exactly Once
Latency	Sub-second	Seconds

Language Options	Java, Clojure, Scala, Python, Ruby	Java, Scala, Python
Development	Use other tool for batch	Batching and streaming are very similar

AsterixDB vs. both:
- Scalable, Fault-Tolerant, and Elastic data ingestion facility
- Plug-and-Play model to cater to wide variety of data sources, through both UDF and Adapters.
- While Storm & Spark are both with custom built solution, e.g, Storm + MongoDB, AsterixDB can reduce the number of moving parts needed to handle Big Data.

Oppgave 7 (20%)

1.

a. Hva er «Bloom Filter» og hva brukes det til? Bruk eksempel til å støtte forklaringen din. (4%)

Svar:

With **a filter,** we want to accept those tuples in the stream that meet a criterion. Accepted tuples are passed to another process as a stream, while other tuples are dropped. A bloom filter is an example of such a filter. A Bloom filter consists of:
1. An array of n bits, initially all 0's.
2. A collection of hash functions h1 , h2 , . . . , hk . Each hash function maps "key" values to n buckets, corresponding to the n bits of the bit-array.
3. A set S of m key values.

The purpose of the Bloom filter is to allow through all stream elements whose keys are in S, while rejecting most of the stream elements whose keys are not in S.
For the example, refer to the lecture notes (slides).

b. Ved å bruke bloom filter-prinsippet fyll ut følgende tabell. Anta at vi skal bruke *h* som hash-funksjon, og at den er definert som følgende *h(x)= y mod 11*, der *y* er hentet henholdsvis fra oddetalls-bits fra *x* eller partalls-bits fra *x*. Eksempel*: $h_1(39) = 011 = 3$, $h_2(39) = 101 = 5$*, osv. (6%)

Strømelement	**Hash-funksjon - h_1**	**Hash-funksjon - h_2**	**Filtrere Innhold** 000 0000 0000
85 = **101 0101** 214 = **1111 1010**	1111 =15 mod 11 = 4 1100=12 mod 11 = 1	000 = 0 1111=4	**1**00 0**1**00 0000 **11** 0100 0000

353 = **01 0110 0011**	11001=25 mod 11 = 3	00101=5	**110 111**0 0001

2. Anta at du skal finne andeler av unike spørringer den siste måneden i et søkesystem fra en strøm av spørringer (search queries). På grunn av plass- og tidsbegrensninger kan du ikke sjekke alle innkomne spørringer og må derfor bruke sampling. Du bestemmer deg for å lese hver 10. spørring (dvs. 10% sampling). Forklar hvorfor dette ikke vil fungerer, dvs. hvorfor vil ikke dette gi korrekt svar? Hva ville være en mer korrekt måte å sample strømmen av spørringene på? (5%)
   Svar:
   A correct answer should at least contain the fact that sampling on every 10th query would mainly focus on the position in stream not the value of the stream element and therefore wouldn't be representative for the set of queries, i.e., the fraction of unique queries in the sample is not equal to the fraction for the stream as a whole. A solution should be focusing on the content, which can be achieved by using hashing on the value, i.e., hash search queries to 10 buckets 0, 1, … , 9. In this way, the sample is equal to all search queries that hash to bucket 0.
   All or none of the instances of a query are selected. Therefore, the fraction of unique queries in the sample is the same as for the stream as a whole.

3. Tenk at du skal bruke sosiale media som Twitter for å analysere trender o.l. Du er spesielt interessert i å vite når et produkt nevnes av folk og hvor ofte de nevnes. Du foreslår å bruke «bucket»-prinsippet til å telle hvor mange ganger dette produktet blir nevnt. Forklar hvordan «bucket»-prinsippet på datastrøm fungerer i dette tilfelle. (5%)
   Svar:
   It is important that the student shows that he/she understand the principle with bit counting. Here to be able to use bucket we have first to convert the problem to bit counting, i.e., for every tweet mentioning a specific product, we set the bit to be 1, 0 otherwise. The bucket is then a segment of the window, represented by a record consisting of:
   - The timestamp of its end [O(log2 N) bits].
   - The number of 1's between its beginning and end.
   - Number of 1's = size of the bucket.

   The constraint on the bucket sizes is that the number of 1's must be a power of 2. Thus, only O(log2 log2 N) bits are required for this count.

Oppgave 8 (15%)

2. Såkalt «*cold start*» er en utfordring når man bruker «collaborative filtering» i et anbefalingssystem. Forklar hva som menes med «cold start» og når dette kan oppstå. Drøft videre hvilke mulige løsninger som finnes for cold start-problemet. (6%)
   Svar:
   Cold-start problem is a problem that occurs when a recommender system is not able to provide recommendation because of a new item, new user, or new community as

explained in the following:
New item problem: Small number of users that rated an item, accurate prediction for this item cannot be generated.
New user problem: Small number of items rated by a user, it is unlikely that there could be an overlap of items rated by this user and active users. User- to-user similarity cannot be reliably computed.
New community problem: Without sufficient ratings, it's hard to differentiate value by personalized CF recommendations.

Possible solutions:
As the solution for new user problem:
– Displaying non-personalized recommendation until the user has rated enough
– Asking the user to describe their taste in aggregate
– Asking the user for demographic information and using ratings of other users with similar demographics as recommendations
As the solution for new item problem:
– Recommending items through non-CF techniques content
analysis or metadata
– Randomly selecting items with few or no ratings and asking user to rate those items.
As the solution for new community problem:
– Provide ratings incentives to a small "bootstrap" subset of the community, before inviting the entire community.

3. Anta følgende bruker-rating tabell.

USERS

PRODUCTS	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	P	3		3			2

Anta videre at du skal bruke «item-item collaborative filtering».
a. Forklar hva er forskjellen(e) mellom «item-item collaborative filtering» og «user-item collaborative filtering». (4%)

Svar:
In this subproblem I specifically invited the candidate to show how they have understood the concepts introduced in the course and reason the answer to the question based on the knowledge they have acquired.

A **user-item filtering** takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked

(focus on similarity between users to recommend items). In contrast, **item-item filtering** will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations (focus on similarity between items based on user ratings to recommend items).

b. Forklar hvordan du vil gå frem for å beregne predikert/estimert verdien av P. Gjør de antakelsene du finner nødvendige. (5%)

Svar: Here we apply item-item filtering as assumed above:

Step 1: Compute the similarities between item #6 and the rest (i.e., sim(6, i), where i = 1,2,3,4,5. Here we could use either cosinus, cosinus centered on 0, or Pearson Correlation to compute similarities.

Step 2: Find ***all positive similarities*** and compute weighted average of ratings to estimate the value of P.

LØSNINGSSKISSE TIL EKSAMENSOPPGAVE I FAG TDT4305 – MAI 2017

NB! Dette er ikkje fullstendige løysingar på oppgåvene, kun skisse med viktige element, hovudsakleg laga for at vi skal ha oversikt over arbeidsmengda på eksamen, og som huskeliste under sensurering. Det er også viktig å være klar over at det også kan vere andre svar ein dei som er gjeve i skissa som vert rekna som korrekt om ein har god grunngjeving.

Oppgåve 1 – Hadoop – 15 % (alle delar tel likt)

a) Blokker av fast størrelse replikert til *n* maskiner, fortrinnsvis på meir enn eitt rack. Viktig poeng at granulariteten er blokk og ikkje fil. Forøvrig, sjå pensum for meir detaljar.
b) Handsamar namnerom til filsystem (filer, katalogar, og metadata for desse).
   Informasjon lagra persistent på lokal disk som to filer: namespace image og edit log.
   Også oversikt for kvar fil kvar blokker er lagra.
   Ikkje nødvendig å lagre dette persistent (kan rekonstruerast frå datanodar ved restart).
c) Data representert som key-value-par
   Map (k1, v1) → List(k2, v2)
   Reduce (k2, List(v2)) → List(k3, v3)
   Viktig å få med at output er liste, mange bommar her.

Oppgåve 2 –Spark – 20 % (alle deler tel likt)

a) val data = sc.textFile("countries.tsv").
b) val data2 = data.map(x => x.split("\t"))
c) val data3 = data2.map(x => (x(0), x(1).split(",")))
d) data3.flatMap(x => x._2).count() eller
   data3.flatMapValues(x => x).count()
e) data3.flatMap(x => x._2).distinct().count()
   /(latMapValues ville ikke virke her siden output er (Land,Term))
f) val data4=data3.flatMapValues(x => x).distinct().map(x => (x._1, 1)).reduceByKey((x,y)=> x+y).collect()
   (Ikkje trekk for manglande collect)

Oppgåve 3 – NoSQL – 5 %

Consistency – er kopiar like?
Availability – alltid tilgjengeleg
Partition tolerance - kan systemet køyre sjølv om det er partisjonert?
CAP-teorem: Ikkje mogleg å tilby alle tre samtidig i eit distribuert system med replikering
Forventa forklaring som viser forståing av termane (og ikkje berre term på C/A/P).

Oppgåve 4 – MinHashing – 10 % (alle deler tel likt)

a) Shingles er n-gram, kontinuerlige subsekvensar av token, som kan brukes for å måle grad av likhet mellom dokument.

b) Svar:

	S1	S2	S3	S4
h1	2	1	1	0
h2	0	0	0	0

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

Anta at du er ansatt i et firma som analyserer interessetrender for det nye kameraet Sony A9 som Amazon nettopp har startet å selge. Firmaet ditt har bestemt seg for å gjøre dette basert på sosiale media data, inkl. Twitter og Facebook.

a) Gå ut fra at du starter med å bestemme deg for finne antall meldinger som nevner ”Sony A9” ved å bruke såkalte stående spørring (”Standing Query”). Forklar hva som er forskjellen(e) mellom ”standing query” og ”ad-hoc query”. Hvordan kan ”standing query” løse oppgaven?

Svar:

- **Ad-hoc queries**: Normale spørringer som kjøres en gang.
  Feks.: Hva er maks. verdi sett så langt?
- **Standing queries**: Spørringer som i prinsippet kjøres på strømmen hele tiden.
  Feks.: Rapporter hver gang man ser en ny maks verdi i datastrømmen.
- For å løse oppgaven kan vi bruke spørring som kjøres på datastrømmen hele tiden slik at for hver ny melding som inneholder ”Sony A9” skal man returnere en ny verdi på S, hvor S er summen av antall meldinger som inneholder ”Sony A9” i datastrømmen.

b) Siden vi kan se sosiale media data som datastrøm har de også de andre karakteristikker og/eller utfordringer enn statiske data. Drøft hva disse er.

Svar: Her kreves det at man lister minst tre karakteristikker samt konsekvensene av disse (utfordringene) for å få full pott.
Karakteristikker:

- Data kommer hele tiden, dvs. uendelig og uten grenser
- Har varierende format og struktur, for eksempel bilder, tekst, etc.
- Kommer i høy hastighet og er dynamisk
- Har stort volum

Utfordringer ligger i å ha metoder for å håndtere disse i form av for eksempel plass (minne og lagringsplass), tid og hastighet (inkl. sanntidsprosessering og skalerbarhet).

c) Som alternativ til ”standing query” velger du ”bit counting” for løse deler av oppgaven. Forklar hvordan kan oppgaven oversettes til ”bit counting”.

Svar: Bit counting kan løse oppgaven ved å oversette alle meldinger til binære tall, hvor en bit settes til 1 dersom meldingen inneholder ”Sony A9”, 0 ellers. Vi reduserer da problemet til å telle enerne i bitstrømmen ved for eksempel å bruke DGIM algoritmen.
NB: For å få full pott holder ikke bare å si at man teller enerne men også nevne hvordan dette gjøres i praksis.

d) Analysen din skal se på trendene de siste 2 ukene og du ser for deg antall ”klikk” og ”kjøp” av produkter i Amazon direkte som en del av analysen. Se for deg at du skal finne ut hvor stor andel av ”klikk” og ”kjøp” er gjort på ”Sony A9”. Til dette formålet velger du nå å bruke glidende-vindu-prinsippet (“sliding window”). Anta at dette vinduet har en størrelse på 500 ”klikk” og ”kjøp” tilsammen. Forklar hvordan du går fram for å beregne denne andelen.

Svar: Her kan man fortsatt bruke bit counting som utgangspunkt. Deretter forventes at man forklarer hvordan man beregner andelen av meldinger som inneholder ”Sony A9” innenfor et vindu med størrelse 500 bits (i/500). Videre er det nødvendig med oppdatering av denne andelen for hver ny som melding kommer inn.

e) Når vi først er inne på ”klikk” drøft hvordan ”bloom filter” kan være nyttig til å finne antall klikk. Gjør de antakelsene du finner nødvendig.

Svar: Bloom filter kan brukes her til å “filtrere” bort gjentakende klikking fra same person, dvs. man kan bruke bloom filter som et steg i tellingen av antall unike klikk. Dvs. først kan man lage et bloom filter ved for eksempel hashe på både URL (linken som klikkes på) og bruker ID (feks. IP-adressen til brukeren). Dette vil hjelpe oss i å identifisere om man har sett klikkingen til brukeren før. Hvis en klikk *ikke er sett* før økes en teller. Ellers lar man telleren være uendret.

Hovedpoenget med spørsmålet er å finne ut om man har forstått prinsippet med Bloom Filter.

f) Bruk “bloom filter”-prinsippet til å fylle ut tabellen nedenfor

Strømelement	**Hash-funksjon - h_1**	**Hash-funksjon - h_2**	**Filtrere Innhold**
			00000000000
56 = 11 1000	110=6	100=4	00001010000
428 = 1 1010 1100	10010=18 mod 11 =7	1110=14 mod 11 =3	00011011000
875 = 11 0110 1011	10111=23 mod 11=1	11001=25 mod 11 =3	01011011000

Hint: bruk *h1(x)=y1 mod 11* og *h2(x)=y2 mod 11* som hash-funksjoner, der verdiene av *y1 og y2* er hentet henholdsvis fra oddetalls-bits fra *x* og partalls-bits fra *x*.

Oppgave 6 – Anbefalingssystem og sosiale grafer (recommender systems and social graphs) – 20 % (alle deler teller likt)

Firmaet du er nyansatt i er spesialist på anbefalingssystemer. Din oppgave er å utvikle gode anbefalingsalgoritmer og metoder.

a) Med fokus på fordeler og ulemper sammenlikn ”collaborative filtering” mot ”content-based recommendation”.

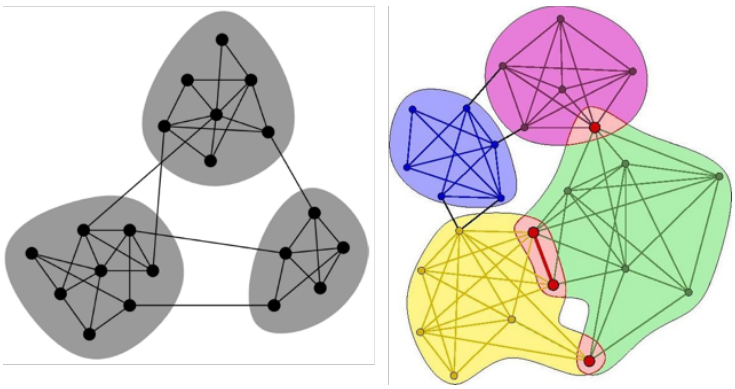
Svar:

Collaborative Filtering	**Content-based Recommender**
Fordeler: - Fungerer på alle type produkt - Lett og intuitivt å sette opp basert bruker/produkt-matrisen	**Fordeler**: - Trenger ikke data på andre brukere - Kan gi anbefaling til brukere med unike smak - Kan anbefale nye og ikke populære produkt (ingen cold start på nye produkter) - Bakgrunnen for anbefaling kan forklares til bruker
Ulemper: - Cold-start-problemet: trenger nok brukere i systemet til å finne ”match” - Sparsitet: veldig få produkt har ratings. Problemer med å finne nok bruker-ratings på produkt - First raters: Kan ikke anbefale produkt som ikke har ratings fra før - Popularitets bias: Kan ikke anbefale produkt til brukere med unike smak. Mest tendens til å anbefale populære produkt.	**Ulemper**: - Avhengig av god feature engineering, i.e., utfordrende med å hente ut features på alt - Anbefaling til nye brukere er avhengig av å kunne ha en god brukerprofil - Overspesialisering, i.e., anbefalingene har tendens til å være kun innen brukerens profil (lav grad av serendipity)

b) Anta at du vil bruke bruker-relasjoner, som feks. “friends” og “followers” fra sosiale media, til å bygge opp brukerprofiler. Slike relasjoner kan tegnes i en graf. Bruk dette som utgangspunkt til å svare på følgende spørsmål:

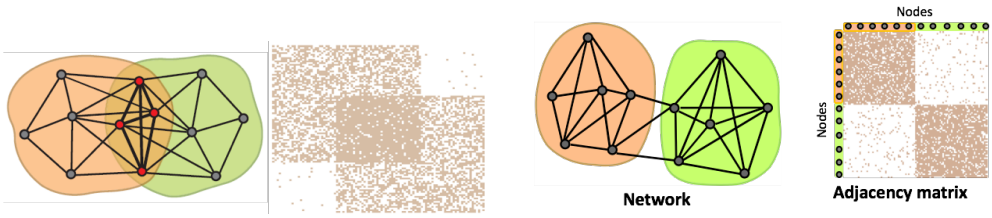
i. Forklar forskjellene på ”*overlapping*” og ”*non-overlapping communities”* i grafer. Hvordan kan vi enkelt finne ut om to grafer overlapper.

Svar:



Overlappende vs. ikke overlappende som vist på figuren over har vi dersom en eller flere noder kan identifiseres til å tilhøre flere “communities”. En enkel måte å finne dette ut på er å bruke

naboskapsmatrise (adjacency matrix). Jo mer kant-tetthet i denne matrisen jo større sannsynliget for overlapp (se figurene nedenfor).



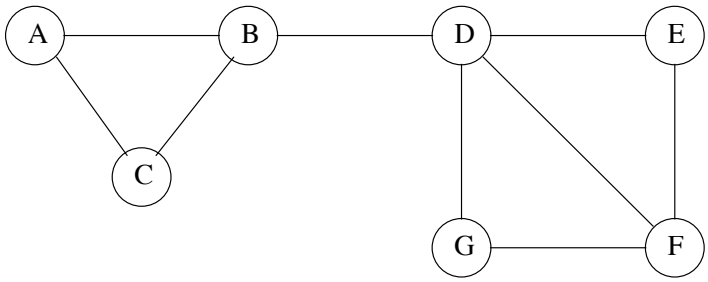
(For å få full uttelling holder ikke bare å nevne at en eller flere noder hører til i flere communities men også hvordan man finner ut dette ved for eksempel overnevnte matrise).

ii. Hva er hovedhensiktene med "community detection"? Forklar.

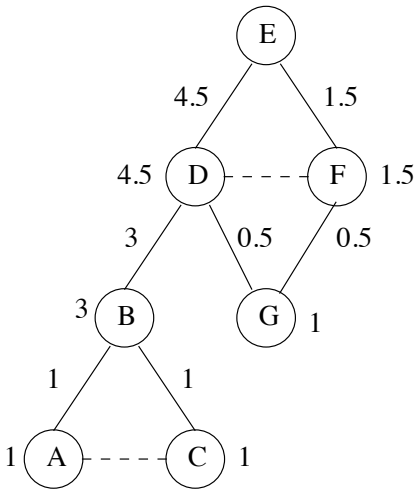
Svar: Hovedhensiktene til "community detection":

- Community detection er en form for graf-clustering som brukes til å oppdage grupperinger og relasjoner mellom grupper.
- Å kunne analysere trender
- Å kunne identifisere grupper som igjen kan brukes til enten markedsføring eller produktanbefaling.

c) Girvan-Newmans metode for beregning av "betweenness" er en vanlig metode innen graf-mining-teori. Vis hvordan du går fra Figur 1 til Figur 2 nedenfor ved å bruke Girvan-Newmans metode til å beregne "*betweenness*" i grafen i Figur 1.



Figur 1: Start-graf

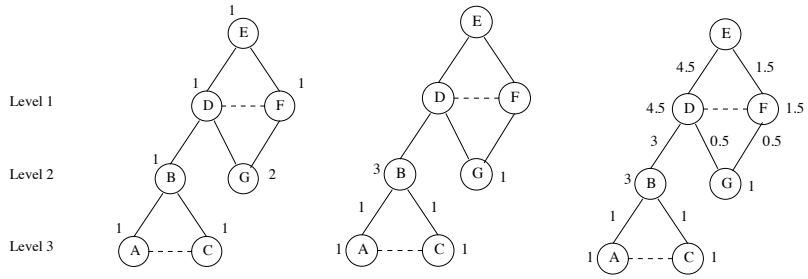


Figur 2: Steg 3 i første iterasjon.

Svar:

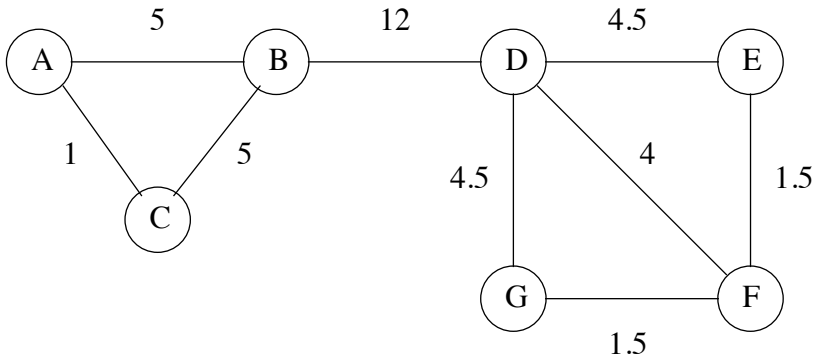
Vi bruker BFS som beskrevet i læreboka.

Gjøre om E til rot-node.

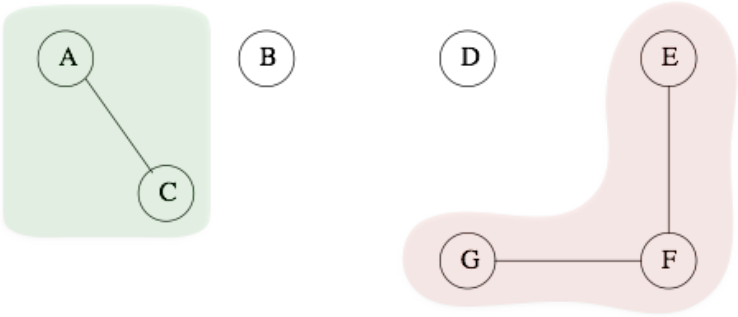


Steg 1 Steg 2 Steg 3

d) Hvordan kan resultatfiguren nedenfor brukes til detektere ”communities”?



Svar:
Vi fjerner alle kanter med **høyste betweeness-verdi** og sitter til slutt igjen med ”communities” som da er komponenter som koplet sammen.



Herfra kan vi også få hierarki av communities.

LØSNINGSSKISSE TIL EKSAMENSOPPGAVE I FAG TDT4305 – MAI 2016

NB! Dette er ikkje fullstendige løysingar på oppgåvene, kun skisse med viktige element, hovudsakleg laga for at vi skal ha oversikt over arbeidsmengda på eksamen, og som huskeliste under sensurering. Det er også viktig å være klar over at det også kan vere andre svar ein dei som er gjeve i skissa som vert rekna som korrekt om ein har god grunngjeving.

Oppgåve 1

Tre viktigste V'ar:

1) *Volume*: enorme mengder data kontinuerlig generert, både brukargenererte og maskingenererte (t.d. sensorar)
   Ikkje i seg sjølv nok til at det er "Big Data"!
2) *Velocity*: hastigheit på datagenerering og prosessering, ofte i sanntid, t.d. umiddelbar deteksjon av trendar på Twitter, ofte straum-data
3) *Variety*: variasjon i typar data, kombinasjon av strukturerte og ustrukturerte

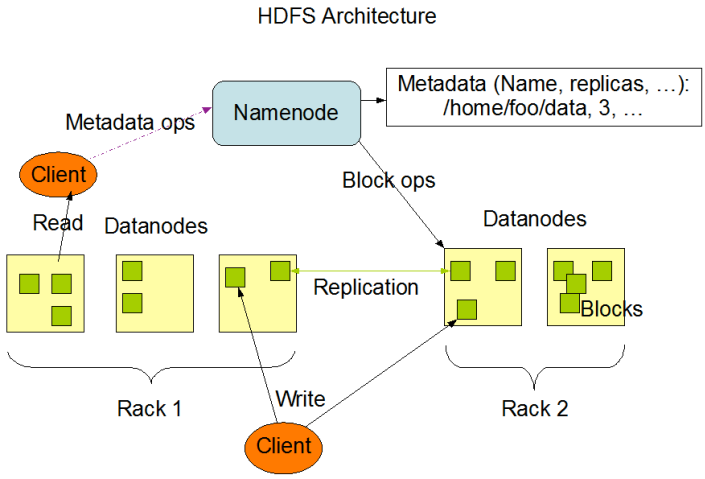
Viktig med forklaring til kvar av dei (særleg på variety er det mange som har tynn forklaring, tildels også på velocity).

Oppgåve 2

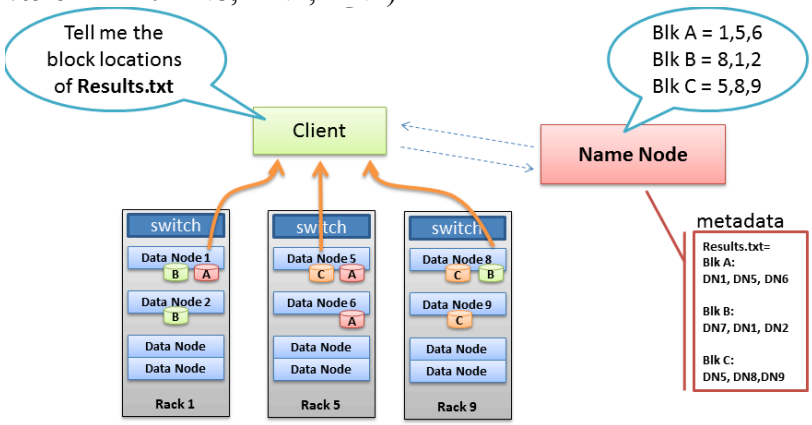
a) Mål:
   1) (Veldig) store filer
   2) Datastraum-aksess
   3) Hyllevare (maskiner)

   Ikkje eigna til:
   1) Data-aksess med krav til liten forseinking (*low-latency data access)*
   2) Mange små filer
   3) Fleire samtidige skrivarar (dvs. er *single-writer*)
   4) Vilkårlege oppdateringar i filer (dvs. er *append-only*)

b) Viktig å få med:
   1) Blokker
   2) Replikering
   3) Klient
   4) Navnenode
   5) Datanoder
   6) Aspekt rundt filer og blokker (nokre misforstår og trur granulariteten er fil og ikkje blokk)



c) Finn blokk-lokasjon(ar) (Klient kontaktar NameNode for å få liste over blokker/noder)
Les fil (blokkar) sekvensielt, velg blokk som er ”nærmast” (liten feil i figuren: på metadata skal det være Blk B: DN8, DN1, DN2)



d) Utføring av applikasjon på YARN:

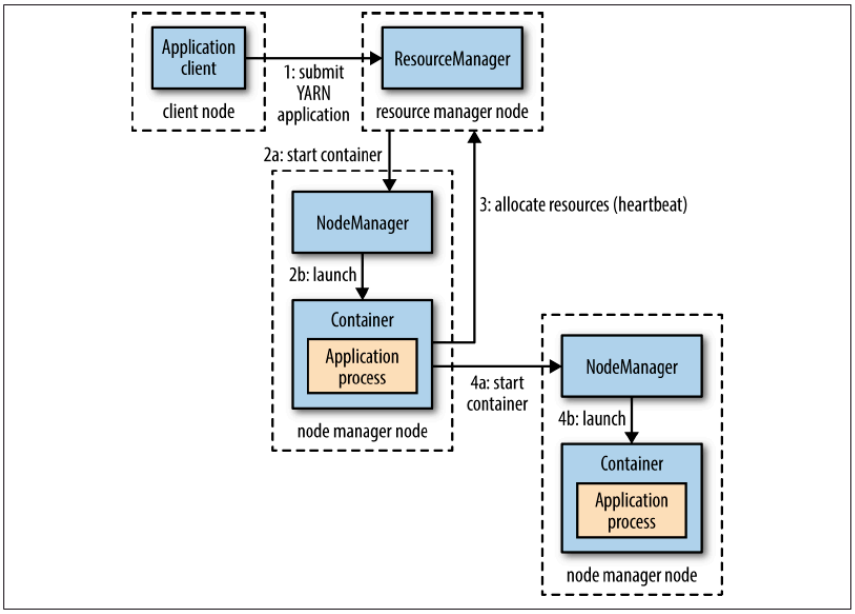


Figure 4-2. How YARN runs an application

Oppgåve 3

a) public void map(key(name), value(age,salary))
write(age,salary)

public void reduce(key, Iterable values)
n = 0
total = 0
foreach val in values
n++
total = total + val
avg = total/n
write(key, avg)

b) Utifrå oppgåveteksta kan vi anta at vi allereie har lese fila inn i ein RDD av par (key,value)=(age,salary), dvs. RDD[(int,int)]. Dette kan gjerast med dei første 3 linjene under (studentane treng altså ikkje ha med desse i svaret):

```
val lines = sc.textFile("PersonInfo.txt")
val records = lines.map(_.split("\t"))
val tuples = records.map(rec => (rec(1).toInt, rec(2).toInt))
```

Mest elegant løysing (Python og Java er også OK):

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

(Det er forventa at dei også har med funksjonen inne i reduceByKey for å vise at dei forstår korleis den vert brukt)

Mindre elegant/effektivt for eksempel:

```
val k = tuples.groupByKey()
val kk = k.mapValues(x => x.toArray.max)
```

Oppgåve 4

a) Det som er prinsippet her er å kunne utnytte sammensatte nøkler. Dvs. vi må ha en tabell med nøkkelen (ExamNo, StudentNo) gjerne sammen med all info vi trenger til å lage resultatet: (ExamNo, StudentNo, CourseName, Grade). Tilsvarende for det andre queriet: (StudentNo, CouseName, StudentName, Grade).

Student(SNo, Name, Email)
Exam(ENo, CourseName, EDay, EMonth, EYear, Duration)
ExamResult(ExamNo, StudentNo, CourseName, Grade)
StudenResult(StudentNo, CourseName, StudentName, Grade)

Også godteke StudentResult(StudentNo,ExamNo...)

b) MongoDB:

- Definerer «shard key» som brukes til å fordele dokumentene mellom tjenere. Dette må være et indeksert felt som finnes i alle dokumenter.
- Range partisjonering eller hash partisjonering
- Tag-aware sharding hvor «shards» kan plasseres i spesielle clusters som f.eks. er geografisk nær.

Hbase:

- Data lagres i «regions» som er sorterte ranges av data som lagres sammen.

- Systemet begynner med en region per tabell. Men tabellen blir splittet automatisk i flere regioner når den blir for stor. Regioner kan også automatisk blitt slått sammen når de blir for små.
- Hver region lagres på en region-server, men en region-server kan lagre mange regioner.

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).

Karakteristikk/Utfordring:
- Data kommer kontinuerlig
- Massiv
- Ubegrenset
- Muligens uendelig og derfor uhåndterlig feks. i forhold til minneforbruk.

Eksempler:
- Data fra sensorer som værstasjoner
- Søketermer fra websøkegrensesnitt
- Datastrøm generert fra klikking på URL'er.

b) Vi skiller mellom to typer spørringer når det gjelder datastrøm. Forklar hva disse er. Bruk eksempler til å støtte forklaringen din.
- **Ad-hoc-spørringer :** Normale spørringer som er kjøres en gang på strømmen.
  Eksempel: Hva er den maksimale verdien sett så langt i strømen *S*?
- **Kontinuerlig/stående søk:** Spørringer som i prinsippet kjøres på strømmen hele tiden.
  Eksempel: Rapporter hver gang en ny maksimumsverdi er sett i en strøm *S*.

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.

Skisse:
- Sett $N = 1000$
- For de første N meldinger regn/tell tweets som inneholder temaet "valg" og "valgkamp", og del på N.
- Deretter for hver ny melding m_i. Hvis m_i inneholder temaet legg/endre snittet med $(1 - j)/ N$, hvor m_j er den eldste meldingen, og j=1 hvis m_j inneholder temaet, 0 ellers.

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

Ja, man kan sette opp en bitstrøm hvor en bit har verdien 1 dersom en melding inneholder temaet og 0 ellers. Deretter kan man telle antall enere i et gitt vindu av størrelse N, 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 mod 11*, der *y* er hentet henholdsvis fra oddetalls-bits fra *x* eller partalls-bits fra *x*.

Eksempel: $h_1(39) = 011 = 3$, $h_2(39) = 101 = 5$, osv.

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.
   - Gjør meldingstrømmen om til bitstrøm som forklart i d)
   - Gjør alle 11 bits til 0
   - Bruk hash-funksjon som i e), og sett bit nr. *h(x)* til 1 for hver input *x*.
   - Hvis alle bits ender med 1 har vi sett strømmen 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.

Her bruker vi samme prinsippet som i e). $h_1(y) = 1101 = 2$, $h_2(y) = 111 = 7$. I meldingen er bit nr. 2 1 og bit nr. 7 er 1, derfor kan vi ha sett strømmen før.

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.
   Finne alle filmer som er mest lik disse til å anbefale neste. Dette kan gjøres ved å finne først hvordan bruker har rated dem, og deretter estimere rating for en ny film basert på en likhetsbergning.

   ii. Ville du brukt innholdsbasert ("content-based") anbefalingsmetode eller "collaborative filtering"? Begrunn svaret ditt.
   Siden selskapet er nytt, har man ikke mange nok brukergrunnlag til å bruke collaborative filtering (for eksempel pga. sparsity). Vi bør derfor bruke content-baset anbefaling hvor vi finner og anbefaler film(er) som er mest lik(e) disse. Her kan vi bruke alle filmattributtene (som sjanger, skuespillere osv.) som grunnlag (for eksempel som features).

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 ratingene 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 rating-ene til film i og j.

Oppgaven dreier seg om Item-Item collaborative filtering, og gjøres som følgende:

1. Regn ut gjennomsnittet av alle brukers ratings for hver film
2. Regn ut likhet mellom ratingene av film nr. 6 og de andre filmene bruker 7 har ratet (dvs. *sim(6, 2) og sim(6, 3)* vha cosinus/pearson-likheten).
3. Bruk sim-verdiene i formell nr.2 til å regne ut estimatet for ratingen. Resultatet blir som følger:

Original Ratings:

	1	2	3	4	5	6	7	8	Avg.
1	1		3			5			3.00
2			5	4			4		4.33
3	2	4		1	2		3		2.40
4		2	4		5			4	3.75
5			4	3	4	2			3.25
6	1		3		3		3.87	2	2.25

Ratings - average ratings for each movie:

	1	2	3	4	5	6	7	8	sim(6, i)
1	-2	0	0	0	0	2	0	0	0.53
2	0	0	0.67	-0.33	0	0	-0.33	0	**0.37**
3	-0.4	1.6	0	-1.4	-0.4	0	0.6	0	**0.05**
4	0	-1.75	0.25	0	1.25	0	0	0.25	0.29
5	0	0	0.75	-0.25	0.75	-1.25	0	0	0.41
6	-1.25	0	0.75	0	0.75	0	0	-0.25	1.00
Svar	3.87471999954169								

r_7_6 = (0.37*4 + 0.05*3)/(0.37+0.05) = **3.87**