Institutt for datateknikk og informasjonsvitenskap
Fakultet for Informasjonsteknologi, matematikk og elektroteknikk
NTNU

TDT4225 Lagring og behandling av store datamengder

Onsdag 9. desember 2009, Kl. 0900-1200, OBS bare tre timer – gjelder ikke senere år.

Oppgaven er utarbeidet av faglærer Kjell Bratbergsengen og kvalitetssikrer Svein-Olaf Hvasshovd

Kontaktperson under eksamen: Kjell Bratbergsengen, telefon 7359 3439 og 906 17 185

Språkform: Bokmål

Tillatte hjelpemidler: D
Ingen trykte eller håndskrevne hjelpemiddel er tillatt.
Bestemt, enkel kalkulator tillatt.

Sensur: 11. januar 2010

Oppgave 1, Diskkontroller (10%)

Beskriv de oppgavene som ligger i en diskkontroller.

Oppgave 2, Lagring av koordinater (30%)

a) Forklar oppbyggingen av et k-d-tre.

b) Forklar oppbyggingen av en gridfil.

Vi har følgende sett av koordinater som skal settes inn (x,y):
(36,91) (3,87) (45,23) (45,21) (98,1) (72,12) (33,76) (45,25) (26,25) (22,50)

c) Sett verdiene over inn i et k-d-tre

d) Sett verdiene over inn i gridfilen hvor blokkstørrelsen er 4. Vis hvordan treet bygges opp og blokker splittes.

e) Til hvilke oppgaver er gridfil egnet?

Oppgave 3, Relasjonsalgebra (30%)

a) Forklar relasjonsalgebraoperasjonen differanse eller minus: **R = A – B.**

b) Forklar hvordan differanse blir utført med gjentatte gjennomløp. Finn også et uttrykk for totalt transportvolum.

c) Forklar hva vi mener med en signatur og hvordan signaturer kan brukes i utførelsen av relasjonsalgebra.

d) Forklar algoritmen i detalj når du utnytter signaturer under utførelse av differanse.

Oppgave 4, System (30%)

Vi er i et land hvor det er en rekke bomveger hvor betalingen skjer ved et køfrisystem. Ved passering av en bomstasjon lagres blant annet brikkenummer, tidspunkt, bomstasjonens identitet, mm. Posten for hver passering tar 100 byte. Fordi kundene kan be om spesifiserte utskrifter, skal data for alle passeringer lagres i minst ett år. Systemet har 80 millioner kunder og 100 millioner biler (brikker). For hver kunde har vi kundens identifikator (64 bit heltall), betalingsmetode, saldo, og betalingshistorikk, til sammen 500 byte per kunde. Disse data er samlet i kunderegistret. For hver brikke har vi brikkens identifikatorer (64 bit heltall), kundenummer til eier av brikken og en status, totalt ca. 60 byte per brikke. Status bestemmes av betalingsform og om brikken er meldt stjålet. For eksempel vil kunder som betaler på forskudd, få gult lys når saldo er under en viss grense, og rød lys når alt er brukt opp. Disse dataene er samlet i brikkeregistret.

Det genereres i gjennomsnitt 100 millioner passeringer hver dag. Prisen per passering avhenger av hvilken bomstasjon som passeres, tid på dagen og kjøretøytype. Kjøretøytypen står oppgitt i brikkeregistret.

Alle data beskrevet over, er lagret i en felles sentralmaskin for hele systemet.

a) Hvor stort er systemets lagerbehov (ett års lagring).

b) Når et kjøretøy passerer har vi bare 0,1 sekund fra brikken blir registrert til passeringslyset (grønt, gult, rødt) skal tennes (72 km/t tilsvarer 20 m/s). Dette er for lite tid til at sentralmaskinen kan involveres og det er en lokal maskin som gjør jobben. Den lokale maskinen kan også samle opp passeringer og sende disse periodisk til sentralmaskinen. Hvilke datastrukturer vil du ha i den lokale maskinen?

c) Sentralmaskinen holder totaloversikten og sender for eksempel ut statusendring til aktuelle brikkenummer. Kravene til forsinkelser er at kundens konto skal oppdateres minst en gang per døgn. Alle passeringer som er eldre enn ett døgn skal altså være reflektert i kundens saldo.
Hvordan vil du organisere registrene i sentralmaskinen: kunderegister og brikkeregister?

d) Hvordan vil du utføre oppdateringene – de som kommer fra de lokale maskinene nevnt i b) – til de sentrale registrene?

Oppgave 1, Diskkontroller (10%), Forslag på løsning

Fysisk styring av disken – roterende maskineri. Posisjonering, start og stopp, …
Formatering, utlegging av sektorer.
Oppdagelse og maskering av ”bad spots”
Optimalisering
SCSI – noen selvstendige søke- og overføringsfunksjoner.

Oppgave 2, Lagring av koordinater. Forslag på løsning

a) Forklar oppbyggingen av et k-d-tre.

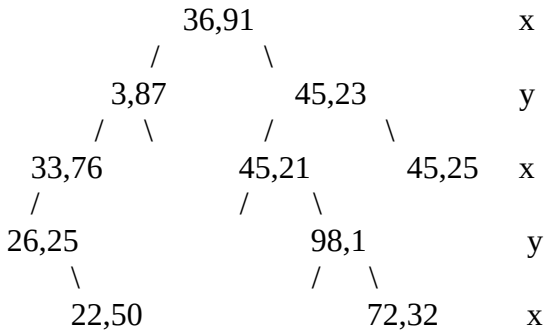
Det er et binærtre. Verdien lagres i noden. En lar deleverdien rotere etter nivå i treet. Dette er en primærlagerstruktur.

b) Forklar oppbyggingen av en gridfil.

Gridfil kan brukes til nøkler med fra to til n dimensjoner. Postene ligger i diskblokker. Det er en n-dimensjonal pekertabell til diskblokkene. Pekertabellen (arrayet) må altså ha like mange dimensjoner som nøkkelen. Vi må også ha like mange vektorer med skilleverdier som det er dimensjoner i nøkkelen. Alle oppslag krever bare en aksess når det er plass til hele pekertabellen i arbeidslager. Når en blokk ikke har plass for post som skal settes inn vil den bli delt. Hvis globale delelinjer allerede går gjennom blokken brukes en av disse, hvis blokken ikke treffes av minst en global delelinje deles blokken i to. Den nye delelinjen blir global. Hvilken dimensjon splittingen skjer i roterer - dermed blir det like mange delelinjer i hver dimensjon (avvik på 1).

Vi har følgende sett av koordinater som skal settes inn (x,y):
(36,91) (3,87) (45,23) (45,21) (98,1) (72,12) (33,76) (45,25) (26,25) (22,50)

c) Sett verdiene over inn i et k-d-tre



d) Sett verdiene over inn i en gridfil hvor blokkstørrelsen er 4. Vi sorterer på den koordinat vi skal dele på. Hvis antall poster per blokk er et oddetall, ligger deleverdien på det midterste tallet. Hvis det er et liketall finer vi deleverdien som midt mellom de to midterste tallene, dvs. deres gjennomsnittsverdi.

A

A	
x	y
3	87
36	91
45	23
45	21

40,5

deleverdi x= 40,5, dvs. verdien midt mellom 36 og 45.

A	B
40,5	

X

A	
x	y
3	87
36	91

B	
x	y
98	1
72	12
45	21
45	23

16,5

Etter første deling og foran ny deling av blokk B, nå i y-retning.

Y

16,5

A	C
A	B
40,5	

X

A	
x	y
3	87
26	*25*
33	*26*
36	91

B	
x	y
98	1
72	12

C	
x	y
45	21
45	23
45	*25*

Etter deling av B og innsetting av tre poster , alle i kursiv.

Y

16,5

D	C
A	B
40,5	

X

A	
x	y

B	
x	y
98	1
72	12

C	
x	y
45	21
45	23
45	25

29,5

D	
x	y
3	87
26	25
33	26
36	91

Etter deling av A som ikke gav noe nytt rom. A måtte deles etter den globale delelinje som allerede gikk gjennom A.

Y

16,5

D	E	C
A	A	B
	29	40

X

A	
x	y

B	
x	y
98	1
72	12

C	
x	y
45	21
45	23
45	25

D	
x	y
3	87
26	25
22	*50*

E	
x	y
33	26
36	91

Etter 2. deling for å få inn siste post, markert med fiolett.

e) Til hvilke oppgaver er gridfil egnet ?

Godt egnet der nøkkelen er en koordinat. Kan være store datamengder til hvert objekt.
Ønskelig at data er jevnest mulig fordelt over hele flaten.

Oppgave 3, Relasjonsalgebra (30%)

a) Forklar relasjonsalgebraoperasjonen differanse eller minus: ***R = A – B.***

A og B må være av samme type, dvs. ha de samme attributter. Dette kravet kan lempes til at A og B minst må ha samme nøkkel. Alle poster i A er i utgangspunktet i R. Fra R strykes alle poster som (nøkkellikhet) også finnes i B. Operandenes rekkefølge er *ikke* likegyldig.

b) *Forklar hvordan differanse blir utført med gjentatte gjennomløp. Finn også et uttrykk for totalt transportvolum.*

Alternativ 1:
Les A poster og fyll opp i WS (working storage – stort M). Les alle B-poster og slett poster i WS som finnes i B. Skriv WS til R. Fortsett lesing av A og fyll opp WS. Les B på nytt og slett poster i WS som også finnes i B. Skrev resten av WS til R. Dette gjentas inntil hele A er lest.

I/O-volum er: $V = V_A + \left\lceil \frac{V_A}{M} \right\rceil V_B + V_R$.

Alternativ 2:
Still nøkler fra B opp i WS inntil WS er full. Les alle poster i A, poster som ikke finner sin make i reservoaret skrives til T. Gjenta inntil alle nøkler i B er plassert i WS. Hvis A allerede er lest, les poster fra T. I siste omgang skrives til R i stedet for til T. A gjennomgår en gradvis tynning.

Postlengden i B er β, Nøkkellengden i begge operander er κ. Antall ganger vi må lese og skrive er A er: $\left\lceil \frac{V_B \kappa}{M\beta} \right\rceil$. Transportvolumet er tilnærmet: $V = 2\left\lceil \frac{V_B \kappa}{M\beta} \right\rceil V_A + V_B$. Hvis alle nøkler fra B rommes i WS vil vi klare oppgaven med en gjennomlesing av begge operander.

Kommentar:

Hvilke metode som skal brukes vurderes ut fra den innbyrdes størrelsen på A og B og hvor stor nøkkelen er i forhold til resten av posten. En liten A og stor nøkkel favoriserer metode 1. En stor B og liten nøkkel i forhold til total postlengde favoriserer alternativ 2. Hva som lønner seg kan regnes ut.

c) *Forklar hva vi mener med en signatur og hvordan signaturer kan brukes i utførelsen av relasjonsalgebra*.

En signatur er en hashverdi av nøkkelen på 2 eller 4 byte. Er to signaturer ulike er også nøklene ulike. Når to signaturer er like, *kan* nøklene være like, men ikke garantert. En må til nøkkelverdiene selv for å avgjøre. I relasjonsalgebra utnyttes det faktum at ulike signaturer må stamme fra ulike nøkler - til å kvitte seg med poster på et tidligst mulig stadium.

d) Forklar algoritmen i detalj når du utnytter signaturer under utførelse av differanse.

Alternativ 1:
Les gjennom B og still signaturer opp i WS. Vi forutsetter at alle signaturer får plass i første omgang. Les alle A, poster med lik signatur skrives til en temporær fil T, alle A-poster uten matchende signatur i WS skrives til R. Alle poster i T stilles opp i WS. B leses på nytt, stryk T(A)-poster i WS som matcher en B-post. Skriv ustrøkne T(A)-poster til R når alle B-poster er lest.

Alternativ 2: Les alle B-poster og sett opp signaturene i B. Les alle A-poster. Signaturer med match merkes. Etter at alle A-poster er lest har vi skrevet ikke-matchende A til R, matchende til T og merket av matchende signaturer i WS. Stryk umerkede signaturer i WS. Les B på nytt, for matchende signaturer stilles også hele nøkkelen opp i WS.
Les alle T-poster. Der nøkkel ikke matcher i WS skriv til R.

Begge metoder har transportvolumet $V = V_A + 2V_B + 2V_T + V_R$

Hvis ikke alle signaturer fra B får plass i WS må vi endre metode. Da kan vi ty til partisjonering.

Metoden uten signatur er best hvis vi har så små operander at det holder med å lese begge operander en gang.

Med signatur må vi regne med å klare større operander før vi får partisjonering. V_T er også meget liten – i området promiller av originaloperanden.

Oppgave 4, System (30%)

Vi er i et land hvor det er en rekke bomveger hvor betalingen skjer ved et køfrisystem. Ved passering av en bomstasjon lagres blant annet brikkenummer, tidspunkt, bomstasjonens identitet, mm. Posten for hver passering tar 100 byte. Fordi kundene kan be om spesifiserte utskrifter, skal data for alle passeringer lagres i minst ett år. Systemet har 80 millioner kunder og 100 millioner biler (brikker). For hver kunde har vi kundens identifikator (64 bit heltall), betalingsmetode, saldo, og betalingshistorikk, til sammen 500 byte per kunde. Disse data er samlet i kunderegistret.

For hver brikke har vi brikkens identifikatorer (64 bit heltall), kundenummer til eier av brikken og en status, totalt ca. 60 byte per brikke. Status bestemmes av betalingsform og om brikken er meldt

stjålet. For eksempel vil kunder som betaler på forskudd, få gult lys når saldo er under en viss grense, og rød lys når alt er brukt opp. Disse dataene er samlet i brikkeregistret.

Det genereres i gjennomsnitt 100 millioner passeringer hver dag. Prisen per passering avhenger av hvilken bomstasjon som passeres, tid på dagen og kjøretøytype. Kjøretøytypen står oppgitt i brikkeregistret.

Alle data beskrevet over, er lagret i en felles sentralmaskin for hele systemet.

a) Hvor stort er systemets lagerbehov (ett års lagring).

Kundedata: 80 000 000 x 500 byte = 40 GB
Brikkedata: 100 000 000 x 60 byte = 6 GB
Passeringsdata 100 000 000 x 366 x 100 = 3660 GB Ta høyde for skuddår.

Passeringsdata er dominerende.

b) Når et kjøretøy passerer har vi bare 0,1 sekund fra brikken blir registrert til passeringslyset (grønt, gult, rødt) skal tennes (72 km/t tilsvarer 20 m/s). Dette er for lite tid til at sentralmaskinen kan involveres og det er en lokal maskin som gjør jobben. Den lokale maskinen kan også samle opp passeringer og sende disse periodisk til sentralmaskinen. Hvilke datastrukturer vil du ha i den lokale maskinen?

Ved hver bom trenger vi en tabell med brikkenummer og status, det vil være nok til å styre lyset. Tabellen bør være en hashtabell med lenket overløp. Blokkfaktor 1, 20 byte per brikke, fyllingsgrad 0,8. Det gir et lagringsbehov på 20x100 000 000/0.8=2.5 GB. Hvis maskinen har 4 GB har vi fortsatt god plass til programmer og opplagrede passeringer. Det trenges plass til en sekvensiell fil for å lagre passeringsdata. Denne filen behøver ikke å være stor da en kan sende data til sentralsystemet så snart en har så mye data at oversendelsen blir effektiv.

Den lokale maskinen trenger egentlig ikke disk bortsett til sikringskopier for brikketabellen. En kan like godt bruke en minnepinne for programmer og sikringskopier.

c) Sentralmaskinen holder totaloversikten og sender for eksempel ut statusendring til aktuelle brikkenummer. Kravene til forsinkelser er at kundens konto skal oppdateres minst en gang per døgn. Alle passeringer som er eldre enn ett døgn skal altså være reflektert i kundens saldo.
Hvordan vil du organisere registrene i sentralmaskinen: kunderegister, brikkeregister og passeringsfil?

Filen med passeringsdata ordnes etter brikkenummer, vi ønsker at passeringer som gjelder samme brikke ligger samlet, men samtidig bør vi også splitte opp datamengden slik at vi ikke skriver alt om igjen ved hver oppdatering. En sekvensiell fil for hver dag, hver uke eller hver måned kan være en løsning. Fra brikkefilen kan vi ha en indeks til hvor disse dataene starter. Det betyr et blokknummer for hver dag, uke eller måned.

En noe mer utfyllende forklaring. Volumet for alle passeringer er meget stort. Det å oppdatere en stor fil er kostbart (tidkrevende) og bør unngås. Etter at dataene er brukt for å oppdatere kundens konto skal ikke dataene nødvendigvis brukes mer. Men de må tas vare på av hensyn til for eksempel revisjon eller klage på faktura. Derfor kan en sortering etter kundenummer være tilstrekkelig. Vi må

gjerne gruppere data for hver dag, men da trenger vi en indeks til blokken i filen hvor kundens data er lagret starter. For hvert år betyr det 366 blokknummer per kunde, skal vi lagre data for mange år blir det fort veldig mye data. Antall kunder er 80 M, et heltall 4 byte per dag per kunde blir 4*80M*366=117,12 GB. Vi benevner dette alternativ a.

Utvilsomt bedre med en fellesindeks for hver fil (dagfil, ukefil eller månedsfil). Registeret for en dag krever 10 GB. Hvis vi bruker en blokkstørrelse på 64 KB blir det 156250 blokker. For hver blokk har vi blokkens største kundenummer som hver tar 8 byte. Indekstabellen for en dag vil ta 8*156250 som er ca. 1,25 MB, på årsbasis blir det 457,5 MB. Alternativ b.

Bruksmessig blir ikke forskjellen så stor. Passeringsdata forventes å bli brukt sjelden når de først er brukt til avregningen. Vi må regne med at alle biler passerer en stasjon de fleste dager i året. Det er lite å spare på bare å lagre dager da en kunde har passering.

Utskriften av alle passeringer for en kunde ett bestemt år, periode eller dag blir: for alternativ a) en diskaksess direkte til blokken der passeringene er lagret for hver dag. For alternativ b) må en slå opp i et indekstre med høyde på 2 pluss datablokk for hver dag. Alternativ er altså 3 ganger så kostbar i bruk.

Registerstørrelsen er 10 GB, 70 GB eller 300 GB henholdsvis for en dag, en uke eller en måned. Kunderegisteret og Brikkeregisteret er begge hashfiler, oppdateres ofte, men få slettinger og nyinnsettinger.

Ved å ha en passeringsfil per dag spares en samfletting med tidligere dagers passeringer. Ved å ha en passeringsfil for et antall dager kan en spare indeksfiler og en får en raskere utskrift de gangene det er aktuelt. Sparing av samfletting ansees så fordelaktig at vi bør velge en passeringsfil per dag.

d) Hvordan vil du utføre oppdateringene – de som kommer fra de lokale maskinene nevnt i b) – til de sentrale registrene?

Passeringsposter gjennomgår en initiell sortering når de kommer til sentralmaskinen. Oppgjøret skjer ved at passeringsfilen sorteres og skrives til en dagfil. Vi har nå valgt å lagre en fil per dag. For hvert brikkenummer summer vi opp passeringsutgiftene og oppdaterer brikkefilen. Hvis brikkefilen er indekssekvensiell kan vi oppdatere etter hvert, da vil oppdateringene komme i nøkkelrekkefølge og oppdateringen er optimal med lite lesing og skriving til disk. Hvis vi bruker en hashbasert fil vil vi legge av oppdateringsposter i en fil. Når vi er ferdig med passeringsfilen vil vi sortere oppdateringsfilen etter hashverdi og oppdatere hashfilen. Vi henter også kundenummer fra hashfilen og kaster en oppdateringspost i en sekvensiell fil. Denne sorteres til slutt på kundenummer, brikkens bidrag til kunden oppdateres og ny status beregnes. Der det er endring vil vi generere en fil med statusoppdateringer. Disse sorteres etter brikkenummer. Brikkeregistret oppdateres og endringsmelding sendes all bomstasjoner.

Kjb/1.12.2009