NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Løsningsforslag for eksamensoppgave i TDT4150 Avanserte databasesystemer

Faglig kontakt under eksamen: Jon Olav Hauglid
Tlf.: 93 80 58 51

Eksamensdato: Fredag 9. juni 2017
Eksamenstid (fra-til): 9.00 – 13.00
Hjelpemiddelkode/Tillatte hjelpemidler: D. Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.
Annen informasjon: Oppgavesettet inneholder 6 oppgaver. Det er angitt i prosent hvor mye hver (del-)oppgave teller ved sensur. Gjør rimelige antagelser der du mener oppgaveteksten er ufullstendig og skriv kort hva du antar. Lykke til!

Målform/språk: Bokmål
Antall sider (uten forside): 2
Antall sider vedlegg: 0

Informasjon om trykking av eksamensoppgave
Originalen er:
1-sidig ☐ **2-sidig** ☑
sort/hvit ☑ **farger** ☐
skal ha flervalgskjema ☐

Kontrollert av:
23.05.17 Svein Erik Bratsberg (sign)
Dato Sign

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

Oppgave 1 – Arkitektur (10 %)

a) «Relational Query Processor» og «Transactional Storage Manager» er to viktige komponenter i et relasjonsdatabasesystem. Forklar kort hva hver av disse to komponentene gjør.

Relational Query Prosessor: Parser SQL, optimaliserer spørringer, lager utføringsplan, utfører SQL statements.
Transactional Storage Manager: Håndterer lagring av data, indekser, buffer, logging og låsing (transaksjoner).

b) Anta at du skal lage en sky-løsning for databaser. Hvilken parallell arkitektur ville du valgt og hvorfor? («Shared Memory», «Shared Disk», «Shared Nothing»).

For en sky-løsning kan man anta mange brukere og lite eller ingen deling av data mellom dem. I tillegg vil det være vanskelig å forutse last, da det ikke er gitt hvor mange brukere man får. Da vil Shared Nothing fungere best. Gir best skalerbarhet og data skal uansett ikke deles. (Alle moderne maskiner er i tillegg Shared Memory pga multicore) Andre løsninger er mulig avhengig av antagelser og prioriteringer.

Oppgave 2 – NoSQL og LSM-trær (20 %)

a) Forklar «consistent hashing» og hvordan det kombineres med replikering av data (project Voldemort).

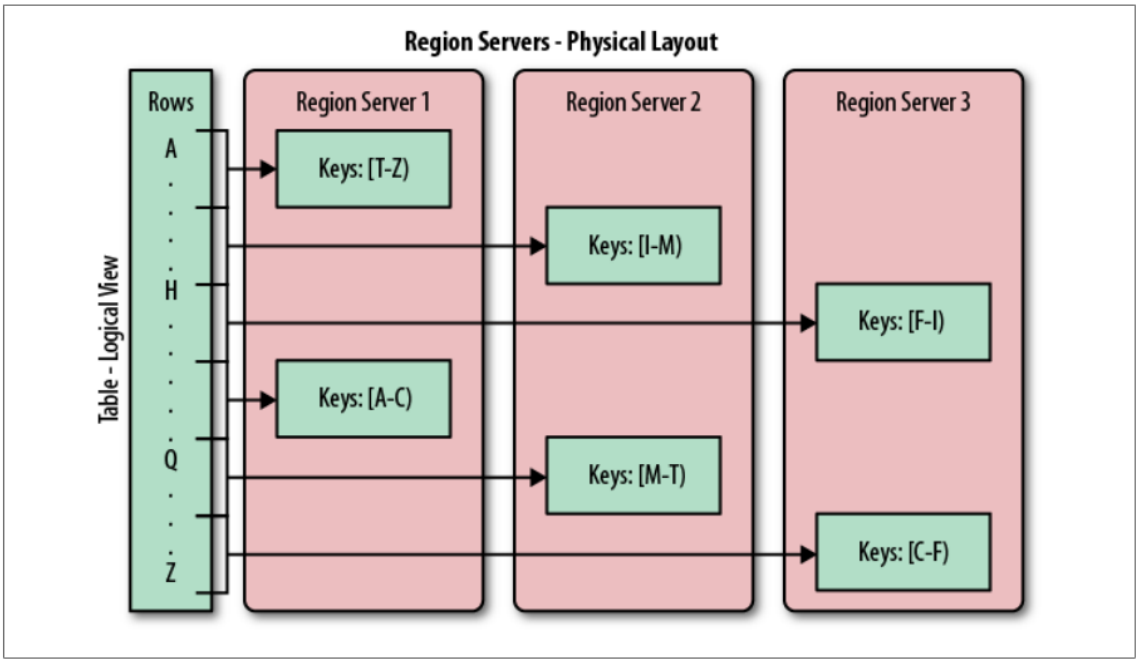
«Consistent hashing» er en metode for å distribuere hashede nøkler mellom forskjellige noder / datamaskiner. Ideen med «consistent hashing» er å tilby en enkel metode for lastbalansering som kombineres med en modell for å håndtere feiltoleranse. Alle noders IP-adresser/portnr (eller noe lignende id-system) hashes inn i en ring, f.eks. med 256 verdier (0-255). Alle nøkler med data hashes også inn i den samme ringen. Nøkkelen med data lagres på den første noden som har en høyere eller lik hashverdi enn nøkkelens hashverdi. Verdiene wrapper rundt ved 255-0. Replikering ordnes ved at en nøkkel og dens tilhørende data også skrives til neste node i ringen, evt, til den nestes neste også (3-vegs-replikering).

b) Hva er dokument-orientert lagring (MongoDB) og hvordan påvirker det skjemamodellering? Gi et eksempel på hvordan Prosjekter og Deltagere («Workers») kan relateres til hverandre i dokument-orientert lagring.

Dokument-orientert lagring betyr at data lagres som en samling av «dokumenter», dvs. som regel data i JSON-format. Opprinnelig oppstod denne modellen for å kunne ha en naturlig lagring av data fra Web-verden. Hvert dokument har en identifikator og en nøstet struktur. Det er i utgangspunktet ikke noen relasjoner mellom dokumenter, men det går an å lagre id'en til et annet dokument.
Det er ikke noe skjema som brukes, så forskjellige dokumenter i samme collection kan ha forskjellige attributter. I dokumentorientert lagring pleier man å se på hva som er hoveddokumentet og så få tilgang til andre dokumenter via dette. Man må da bestemme seg for en nøstet datastruktur. Figur 24.1 i Elmasri&Navathe viser et eksempel med Projects som nøster i seg Workers. Her blir det en dobbeltlagring av Workers hvis en Worker er knyttet til flere Projects. Redundans er en mulig konsekvens av dokumentorientert lagring.

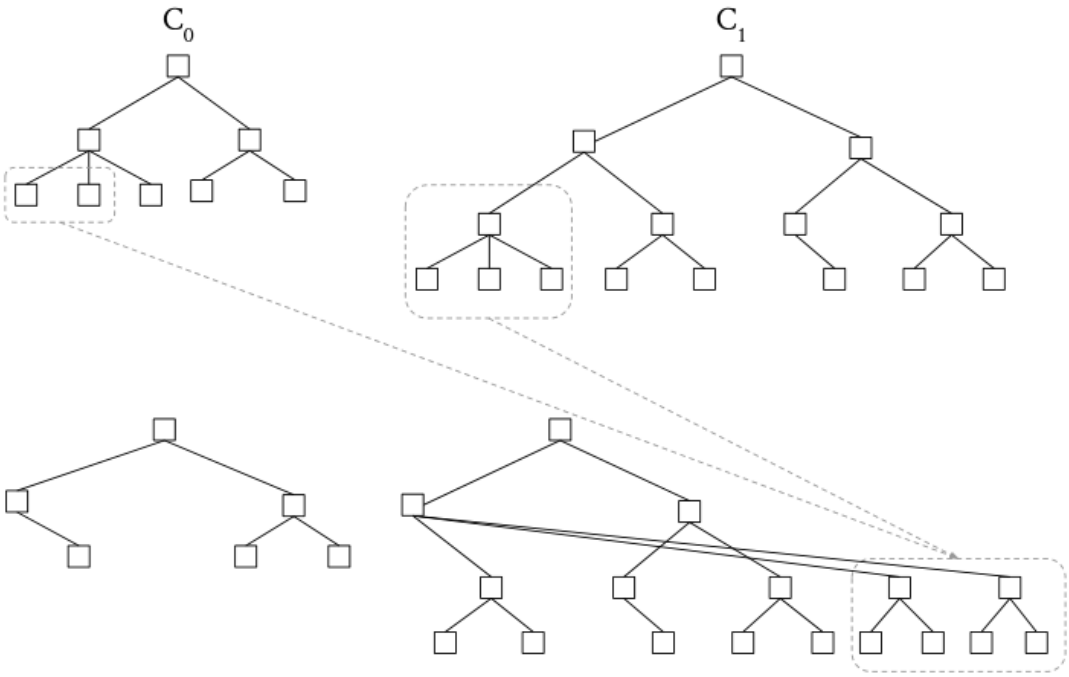
c) Forklar hvordan regioner («regions») og regionservere i Apache HBase fungerer når mengden av data øker.

Enheten for skalerbarhet og lastbalansering i HBase kalles en region. Regioner er stort sett kontinuerlige områder av sorterte rader som lagres sammen. De blir dynamisk splittet av systemet når størrelsen blir for stor. De kan også bli flettet sammen for å redusere antallet.. I utgangspunktet er det bare en region i en tabell, men så snart nye data legges til, vil systemet overvåke for å se at regionen ikke blir større en forhåndsdefinert maksimal størrelse. Hvis den blir stor nok, blir regionen delt i to på den midterste nøkkelen. Hver region er håndtert av en regiontjener (server), men hver av tjenerne kan håndtere mange regioner.



d) Forklar begrepet «rolling merge» som er brukt i LSM-trær («log-structured merge trees»).

Figuren under viser et eksempel på et steg i en "rullende fletting" (rolling merge). Her blir poster fra C_0 flettet inn i C_1 . Prosessen allokerer først det den kaller "den tømmende blokk" (emptying block) som blir et buffer for data fra C_0, og så allokerer den ei blokk som kalles "innblokka» (the filling block) hvor flettede poster fra C_0 og C_1 blir satt inn. Sorteringen av postene fra C_0 og C_1 blir ivaretatt. Når innblokka er full, blir den lagt til enden av C_1, og ei ny innblokk blir allokert. Prosessen kjører til alle postene fra C_0 er flettet inn i C_1.



Oppgave 3 – Spørreoptimalisering (30 %)

a) Anta at join skal utføres på tabellene A og B. Tabell A har T_A tupler og tabell B har T_B tupler. Hvor kostbart vil det være å gjøre dette med Nested Loop Join? Hva med Merge Join? Beskriv eventuelle antagelser du gjør.

*Nested Loop: Antar A som ytre og B som indre. Må lese Ta tupler og for hver av disse lese Tb. Dermed Ta + Ta * Tb. Antar ubetydelig proesseringstid.*
*Merge Join. Hvis A og B begge er sortert, kan man lese A og B bare en gang – mao. Ta + Tb. Hvis A og B ikke er sortert, er det Ta * Log Ta + Tb * Log Tb i tillegg.*

b) Det kan være vanskelig for en spørreoptimalisator å behandle en spørring der tre tabeller joines på samme måte som en spørring der 30 tabeller skal joines. Forklar hvorfor og forklar kort hva som må gjøres forskjellig.

Når 3 tabeller skal joines, er det enkelt å vurdere alle joinrekkefølger (3x2x1). Det er det ikke når 30 tabeller skal joines (30x29x28x...). Det er rett og slett for mange mulige rekkefølger. Dermed må man bruke heuristikker. F.eks. joine de minste tabellene først for å forhåpentligvis minimere størrelsen på mellomresultater.

c) En blokkerende relasjonsoperator er en operator som krever at en eller flere operander er fullstendige før operatoren kan utføres (dvs. kan ikke utføres på delresultat). Gi to eksempler på ikke-blokkerende operatorer og to eksempler på blokkerende operatorer.

Blokkerende: Sortering, Gruppering, Join (untatt f.eks. merge join der begge operander er sortert).
Ikke blokkerende: Seleksjon og projeksjon. Merge join med begge operander sortert.

d) Nevn minst tre grunner til at en spørring kan ta lang tid å utføre.

Stor datamenge, manglende indekser, dårlig utføringsplan, treg maskinvare, komplisert spørring, blokkering pga låser.

Oppgave 4 – Distribuerte databasesystemer (15 %)

a) Hva er replikering og hvorfor kan det være ønskelig?

Identiske kopier av tabeller/partisjoner lagret på forskjellig sted.
Ønskelig pga redundans (og feiltoleranse), økt ytelse ved utføring av spørringer da forskjellige spørringer kan lese fra forskjellige replikat (lastdeling).

b) En transaksjon ønsker å først opppdatere rad X og deretter rad Y. En annen transaksjon ønsker å oppdatere rad Y først og deretter rad X. Forklar hvorfor dette kan gi vranglås ved synkron gruppereplikering. Hva skjer i det samme tilfellet hvis asynkron gruppereplikering blir brukt?

Først låser transaksjon T1 X på node N1 og N2. Så låser T2 Y på N2 og N1. Deretter prøver T1 å låse Y og T2 å låse X. Da vil T1 og T2 vente på hverandre og vi har vranglås.
Ved asynkron gruppereplikering vil T1 låse X på N1, T2 låse Y på N2, T1 låse Y på N1, T2 låse X på N2. Alle låser tas og transaksjonene committer. Men under oppdatering av nodene etter at transaksjonene er ferdige, vil vranglåsen bli funnet og en av transaksjonene må rulles tilbake. Dette byter ACID siden transaksjonene alt har committet.

Oppgave 5 – Rangering og skyline (10 %)

a) Anta en tabell med hybler som inneholder pris og avstand til Gløshaugen. Lag et eksempel som viser at Skyline-operatoren brukt på denne tabellen kan gi mer enn to hybler som svar.

(Avstand, Pris) = (1 km, 8000 kr), (2 km, 6000 kr), (5 km, 5000 kr). Den nærmeste og den billigste vil alltid være med. Men (2 km, 6000 kr) er også med fordi den er billigere enn (1 km, 8000 kr) og samtidig nærmere enn (5 km, 5000 kr).

b) Krever Rank Join at begge operandene er sortert først? Begrunn svaret.

Ja. Poenget med Rank Join er at man stopper utføringen av joinet når man er sikker på at man har fått de tuplene man trenger for å svare på en top-k spørring. For å være sikker på det, må tuplene være sortert på de kolonnene som brukes i top-k uttrykket.

Oppgave 6 – Tema fra seminarartikler (15 %)

a) Fungerer SharedDBs løsning for spørreoptimalisering best ved høy eller lav last? Begrunn svaret.

SharedDB baserer seg på å optimalisere på tvers av spørringer slik at subspørringer som er felles mellom flere spørringer, bare blir utført en gang. Hvis det er lav last (få spørringer), får man enten lite effekt av denne optimaliseringen eller man må vente lengre for å få flere spørringer å optimalisere over (dette introduserer forsinkelser). SharedDB fungerer derfor best ved høy last (mange samtidige spørringer), det den totale mengden arbeid utført kan bli redusert mest i forhold til tradisjonell utføring av spørringer uavhengig av hverandre.

b) Forklar kort hvordan NoDB klarer å aksessere data i rådatafiler effektivt.

NoDB brygger gradvis opp en indeks (positional map) over rådatafiler etter hvert som den utfører spørringer. Dermed blir oppslag i filene mer og mer effektivt.

c) Anta en tabell T med en kolonne C der tuplene i utgangspunktet er i tilfeldig rekkefølge. Ved bruk av Database Cracking, hva skjer hvis det kommer en spørring med betingelsen «T.C > 10»? Hvis det etterpå kommer en spørring med «T.C < 8» og en spørring med «T.C > 15», hva skjer hvis det til slutt kommer en spørring med «T.C > 8 AND T.C < 12»?

Først blir tuplene sortert i to grupper. T.C > 10 og T.C <= 10. All data må aksesseres for å gjøre dette.
Deretter blir T.C <= 10 splittet i T.C < 8 og (T.C >= 8 og T.C <= 10).
Deretter blir T.C > 10 splittet i T.C > 15 og (T.C <= 15 og T.C > 10).
Når siste spørring kommer, holder det å aksessere (T.C >= 8 og T.C <= 10) + (T.C <= 15 og T.C > 10). Man trenger mao. bare å aksessere en del av dataene og ikke alle data slik som ved første spørring. Gruppe (T.C >= 8 og T.C <= 10) blir så splittet i T.C = 8 og (T.C > 8 og T.C <= 10). Gruppe (T.C <= 15 og T.C > 10) blir splittet i (T.C <12 og T.C > 10) og (T.C <= 15 og T.C >= 12).