NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

Løsningsforslag Eksamen i TDT4190 Distribuerte systemer

Faglig kontakt under eksamen: Norvald Ryeng
Tlf.: 97 17 49 80

Eksamensdato: Fredag 6. juni 2014
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 8 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: 3
Antall sider vedlegg: 0

Kontrollert av:

_____

Dato Sign

Oppgave 1 – Distribuerte objekter og fjernkall (10 %)

a) Hvorfor er det enklere for en tjener å tilby kall-semantikken minst-én-gang (at-least-once) i stedet for maksimalt-én-gang (at-most-once)?

For å tilby maksimalt-én-gang må tjeneren holde oversikt over hvilke forespørsler som er mottatt og hvilke svar som er gitt slik at den kan svare på gjentatte forespørsel-meldinger uten å utføre operasjonen på nytt.

b) Hva blir konsekvensene når en svarmelding forsvinner i et system som bruker kall-semantikken:
   1. minst-én-gang (at-least-once)?
   2. maksimalt-én-gang (at-most-once)?

Begrunn svaret.

1. Meldingsgang: Klienten sender en forespørsel, tjeneren utfører operasjonen og sender en svarmelding tilbake til klienten. Svarmeldingen forsvinner. Klienten fortsetter å vente på svar inntil den timer ut etter en fastsatt venteperiode. Etter timeout vil klienten på nytt sende forespørselen til tjeneren, og tjeneren vil utføre operasjonen på nytt og sende svaret på nytt. Hvis også dette svaret forsvinner, gjentas hele runden, og slik fortsetter det til klienten etter et visst antall forsøk gir opp og feiler. Hvis svaret til slutt går bra, fortsetter klienten eksekveringen.

Konsekvens: Operasjonen blir utført minst to ganger, muligens flere.

2. Meldingsgang: Klienten sender en forespørsel, tjeneren utfører operasjonen, sender en svarmelding tilbake til klienten og lagrer svaret. Svarmeldingen forsvinner. Klienten fortsetter å vente på svar inntil den timer ut etter en fastsatt venteperiode. Etter timeout vil klienten på nytt sende forespørselen på tjeneren. Tjeneren vil oppdage at dette er en forespørsel den har utført før og sende det svaret den lagret forrige gang. Hvis også dette svaret forsvinner, gjentas hele runden med ny forespørsel og ny utsendelse av det lagrede svaret. Slik fortsetter det til klienten etter et visst antall forsøk gir opp og feiler, eller at det til slutt går bra og klienten fortsetter eksekveringen.

Hvis klienten til slutt mottar svaret, kan den sende en bekreftelsesmelding til tjeneren slik at denne kan slette det lagrede svaret. Dette er ikke nødvendig for at systemet skal fungere korrekt, men vil redusere behovet for å huske svar på tjenersiden.

Konsekvens: Operasjonen blir utført bare én gang.

Oppgave 2 – Likemannsnettverk («Peer-to-peer networks») (5 % på a, 10 % på b)

a) I Pastry brukes en hashfunksjon til å beregne ID for både noder og objekter. Hvorfor er det en fordel om denne hashfunksjonen gir ID-er med jevn spredning over hele verdiområdet?

ID-er angir hvor i Pastry-ringen noder blir plassert og hvor objekter blir lagret. Hvis spredningen er veldig ujevn, kan det blir dårlig lastbalansering – mange objekter kan bli hashet til en og samme node mens andre noder kan ende opp helt uten objekter.
Ujevn spredning fører også til at mange noder vil få samme ID-prefiks. Det vil føre til at mange noder vil passe til samme rute i prefiksrutingtabellene og at mange ruter ikke vil ha noen noder med passende prefiks. Det vil kunne gi mindre optimal ruting.
Jevn spredning reduserer også risikoen for kollisjoner – dvs. at to noder (eller objekter) blir tildelt samme verdi.

b) I utgangspunktet lagres objekter i Pastry bare på en node slik at objekter går tapt hvis en node dør. Hvordan ville du ha utvidet Pastry slik at man unngår tap når en node dør? Prøv å lage en løsning som er så effektiv som mulig.

For å unngå «single point of failure», kan vi lagre hvert objekt på mer enn en node. En mulighet å bruke to hashfunksjoner som forhåpentligvis gir forskjellige ID-er. Ved oppslag prøver man først den ene hashfunksjonen og hvis ingen svar den andre hashfunksjonen.
En annen, og bedre mulighet, er å la node B lagre kopier av alle objekter på node A, der B er den noden som vil få ansvaret for As ID-er hvis A dør. Dermed vil objektene automatisk fortsatt være tilgjengelig hvis A dør.
I begge tilfeller må nye kopier lages når en node dør for å opprettholde redundansen.
Det finnes også andre mulige løsninger.

Oppgave 3 – Sikkerhet (15 %)

a) Hvordan kan man beskytte seg mot avlytting («eavesdropping») av meldinger?

Man kan kryptere meldingene.

b) Hvordan kan man beskytte seg mot resending («replaying») av meldinger?

Ved å generere et tilfeldig tall, nonce, som brukes kun for et par av forespørsel-svar meldinger. Sesjonsnøkler er også en mulighet. De vil ha gyldighet for flere par av forespørsler.

c) Hva er «chosen-plaintext attack» og hvordan kan man beskytte seg mot det?

Hvis meldingene er mindre enn nøkkelen, kan man prøve et «brute force» angrep på meldingen istedenfor nøkkelen. Dette er aktuelt ved asymmetriske algoritmer der den offentlige nøkkelen brukt for kryptering er kjent. Man kan generere tilfeldige meldinger, kryptere dem med den offentlige nøkkelen og sammenligne resultatet med meldingen man har snappet opp. Unngås ved å lage større meldinger (f.eks. vha. padding).

Oppgave 4 – Distribuerte filsystemer (15 %)

I en bedrift som bruker NFS (Network File System) krasjer tjeneren plutselig.

a) Hvilke operasjoner kan klientene gjøre mens tjeneren er nede? Begrunn svaret.

I en veldig begrenset periode, så lenge cacheinnholdet er ferskere enn grensen ("freshness interval"), typisk 3-30 sekunder for filer og 30-60 sekunder for kataloger, kan klientapplikasjoner lese de blokkene som er cachet på klienten. Dette går fordi protokollen forsøker å begrense meldingssendingen ved å ikke alltid sjekke at cachen er helt oppdatert.

Applikasjonene kan også skrive en stund, men blokkene blir bare bufret på klienten. På et tidspunkt vil klienten finne ut at den skal flushe blokkene, og da vil den ikke klare å få kontakt med tjeneren.

Til slutt vil klienten henge og ikke kunne utføre noen operasjoner. Årsaken er at NFS kun opererer på blokker og ikke cacher så mye på klienten. NFS har heller ikke noe close-kall, så operasjoner går direkte mot tjeneren hele tiden.

Hvis klienten har softmountet NFS-filsystemet, vil filoperasjonene til slutt feile. Hvis filsystemet er hardmountet vil filoperasjonene henge helt til tjeneren kommre opp igjen.

b) Hvilke operasjoner kunne klientene gjort dersom bedriften hadde brukt AFS (Andrew File System)? Begrunn svaret.

Klientapplikasjoner kan fortsette å lese og skrive filer som er cachet lokalt. AFS cacher hele filer, så applikasjoner som allerede har åpnet alle filene sine kan jobbe i fred og ro. Men de kan ikke lukke filene, da vil kallet henge til tjeneren kommer opp igjen.

Dette er mulig fordi AFS cacher hele filer og generelt veldig mye data. Synkronisering mellom klient og tjener skjer ved open og close, så lesing og skriving blir ikke berørt av at tjeneren krasjer. Lesing og skriving mellom open og close går direkte til lokal disk på klienten.

I noen tilfeller kan klientapplikasjoner også åpne filer uten å kontakte tjeneren. Dette skjer hvis klienten har et gyldig callback promise for den aktuelle filen. Disse vil time ut etter en stund, så det vil bare fungere i en liten periode.

Oppgave 5 – Tid og global tilstand (15 %)

a) Hvorfor kan man ikke nødvendigvis bruke fysisk tid til å utlede rekkefølge mellom hendelser i et distribuert system?

Fordi fysisk tid leses fra lokal klokke og det er slettes ikke sikkert at to forskjellige klokker viser det samme («skew»). Og forskjellen kan øke over tid («drift»). Selv med kontinuerlig synkronisering er det umulig å få klokkene helt eksakt like siden man ikke vet helt nøyaktig hvor lang tid en sykroniseringsmelding bruker fra en node til en annen.

b) En av måtene klokker blir synkronisert på i NTP, er symmetrisk modus der to tjenere kontinuerlig sender par av meldinger seg i mellom. Hva må til for at denne synkroniseringen skal bli så nøyaktig som mulig?

Maksimal feil er ±d/2. Der d er summen av overføringtiden til et meldingspar. Antagelsen som ligger i bunn er at meldingene i et meldingspar hadde identisk overføringstid. Maksimalt feil får man dermed hvis den ene meldingen tok all tid. For å få så liten feil som mulig er det derfor viktig at summen av overføringstiden er så liten som mulig. (Selvsagt er det ideelt om begge meldingene har identisk overføringstid, men det kan man ikke vite – total overføringstid kan man derimot vite)

c) Anta et globalt tilstandspredikat ϕ og et distribuert system med flere prosesser. For en gitt global historie kan vi ikke nødvendigvis si om ϕ har vært oppfylt eller ikke. Lag et enkelt eksempel som illustrerer dette.

S00 => S10 eller S01
S10 => S11
S01 => S11
Predikat bare sant I S10. Men vi kan ha gått enten S00 => S10 => S11 eller S00 => S01 => S11. Dermed har vi «possibly» og ikke «definitly».

Oppgave 6 – Koordinering og enighet (10 %)

Kan man implementere kausal ordning av multicastmeldinger ved hjelp av Lamports logiske klokke, eller er det nødvendig med vektorklokker? Begrunn svaret.

Nei, Lamports logiske klokke (en teller hos hver node) er ikke nok til å utlede «skjedde før» relasjoner og dermed kunne gi kausal ordning av meldinger. For eksempel kan man tenke seg et system med kun lokale hendelser. Da vil ingen hendelser hos en node ha «skjedd før» hendelser hos andre noder, men en node vil kunne ha klokkeverdier som både er mindre, lik og høyere enn klokkeverdier hos andre noder.
Man trenger vektorklokker (en teller per node hos hver node) som i praksis holder kontroll med hvor mange hendelser hos andre noder, en gitt hendelse har hatt mulighet til å bli påvirket av. Dermed kan man utlede kausalitet.

Oppgave 7 – Distribuerte transaksjoner (10 %)

Anta at vi har et distribuert databasesystem som bruker strikt tofaselåsing. Det finnes bare en låsetype, eksklusiv lås. Det er tre noder i systemet: n1, n2 og n3. Hver node har en tabell: t1 på n1, t2 på n2 og t3 på n3. Hver node utfører en og en operasjon (venter til én operasjon er ferdig før neste startes). Låser er distribuerte – hver node har informasjon om låser på de tabeller noden lagrer.

Anta operasjoner i følgende rekkefølge:

- n1: START TRANSACTION (kall denne trans1)
- n2: START TRANSACTION (kall denne trans2)
- n3: START TRANSACTION (kall denne trans3)
- n1: INSERT INTO t1 …
- n3: INSERT INTO t2 …
- n2: INSERT INTO t3 …
- n2: INSERT INTO t2 …
- n1: INSERT INTO t1 …
- n3: INSERT INTO t1 …
- n1: INSERT INTO t3 …

Vis steg-for-steg hvordan edge-chasing kan finne denne vranglåsen.

1: n1 utfører trans1 og sender melding til n3 for å få låst t3.
2: n3 ser at t3 er låst av trans2 og kontakter n2.
3: n2 ser at trans2 venter på lås på t2 og kontakter n2.
4: n2 ser at t2 er låst av trans3 og kontakter n3.
5: n3 ser at trans3 venter på lås på t1 og kontakter n1.
6: n1 ser at den får meldingen i retur => vi har oppdaget vranglås.

Oppgave 8 – Google case study (10 %)

a) Hvilke oppgaver har masteren i GFS (Google File System)?

Masteren holder orden på metadata – hvilke filer finnes, hvilke chunks består de av og hvor er de lagret. Aksesskontroll. Organiserer plassering av chunks mhp replikering. Logger til disk (replikert) for pålitelighet.

b) Forklar hvordan Chubby brukes til å implementere monitorering av Bigtable-tjenere.

Hver tablet-tjener lager en fil i Chubby og tar eksklusiv lås på den. At en fil finnes (og er låst) betyr at tjeneren er klar og kan bli tildelt tablets av master. Masteren prøver hele tiden å få lås på filene. Hvis den klarer det, betyr det at tjeneren er nede (har ikke kontakt med Chubby). Da sletter master filen. Hvis tablet-tjeneren finner at filen dens er borte, betyr det at den har hatt nettverksproblemer og at den ikke kan fortsette uten å introdusere konsistensproblemer (siden den kan ha gått glipp av oppdateringer).