To-fase commit – Nettverskprogrammering

Jonas Brager Jacobsen

21.04.2020

Innhold

[Introduksjon 3](#_Toc38288993)

[Rask innføring i to-fase commit 4](#_Toc38288994)

[Oversikt over viktige klasser 5](#_Toc38288995)

[Valg av løsninger og implementert funksjonalitet 7](#_Toc38288996)

[Kommunikasjon 7](#_Toc38288997)

[Tilstander og tilstandslagring 7](#_Toc38288998)

[Fremtidig arbeid med oversikt over mangler og mulige forbedringer 8](#_Toc38288999)

[Eksempler som viser bruk av løsningen 8](#_Toc38289000)

[Installasjonsinstrukser 9](#_Toc38289001)

[Kjøre prosjektet 9](#_Toc38289002)

[Lenke til kode-repository 10](#_Toc38289003)

[Referanser 10](#_Toc38289004)

# Introduksjon

Denne oppgaven er en frivillig prosjektoppgave i faget TDAT2004 på NTNU. Oppgaven går ut på å sette seg inn i nødvendig teori for to-fase commit, og så implementere sin egen versjon som illustrerer slike distribuerte transaksjoner, med eksempler på transaksjoner som fullfører og som feiler.

Løsningen min av distribuerte transaksjoner ved to-fase commit er skrevet i Java. Dette er fordi det er et språk som har gode innebygde biblioteker for operasjoner som må gjøres underveis, i tillegg til at det er et språk jeg behersker bedre enn ander alternativer. De ulike delene av programmet bruker Websockets for å kommunisere.

# Rask innføring i to-fase commit

# Oversikt over viktige klasser

* Client

Abstrakt klasse som arves av TaskManager og ResourceManager. Denne klassen håndterer sending av meldinger, og oppretter en egen tråd for å lytte etter- og ta imot innkommende meldinger. Klasser som arver Client må implementere sin egen måte metode for å håndtere innkommende meldinger.

Client implementerer Runnable, som gjør at de kan kjøres i hver sin tråd. Dette er kjekt under utvikling for å kunne starte alle instanser fra samme program.

* Message

Denne klassen er meldingene som blir sendt mellom klientene. En melding har en transaksjons-id, en klient-id (ID-en til ResourceManager, ikke avsender), en meldingstype som f.eks COMMIT eller ROLLBACK, og har mulighet for å inneholde data.

* TaskManager

Arver Client. Har som hovedoppgave å administrere transaksjoner. Mottar en transaksjon fra utsiden, som den distribuerer til alle ResourceManagers, og håndterer de ulike svarene fra ResourceManagers i løpet av prosessen ved to-fase commit.

TaskManager holder styr på hvilken tilstand transaksjonen er i til enhver tid, og lagrer den persistent fortløpende. TaskManager logger også alt som skjer i en tekstfil.

* TM\_State

Klasse som brukes av TaskManager for å ha oversikt over tilstanden til en transaksjon. TM\_State inneholder en tranaksjons-id, transaksjons-dataen, og en liste med tilstanden til TaskManageren og alle ResourceManagers fra TaskManagerens perspektiv.

* ResourceManager

Arver Client. Mottar transaksjoner fra TaskManager som den er ment å lagre i sin database, representert av en tekstfil. Svarer TaskManager når den ber om at transaksjonen skal gjøres klar, committes, rulles tilbake, eller reverseres.

I likhet med TaskManager lagrer ResourceManager sin egen tilstand og logg.

* RM\_State

Klasse som brukes av ResourceManager for å ha oversikt over tilstanden til en transaksjon fra eget perspektiv. RM\_State inneholder en transaksjons-id, transaksjons-dataen, og ResourceManageren sin tilstand.

* Main

Denne klassen brukes for å teste systemet. Den oppretter instanser av TaskManager og ResourceManagers, og kjører noen transaksjoner.

Legg ved klassediagram

# Valg av løsninger og implementert funksjonalitet

## Kommunikasjon

Et krav jeg satte til programmet var at det skulle kunne håndtere meldinger asynkront, og da ønsket jeg å bruke en kommunikasjonsteknologi som håndterer det bra. Valget falt på Websockets som tillater server og klient å sende hverandre meldinger på et hvilket som helst tidspunkt uten at noen har sent en forespørsel først. Skulle jeg gjort det samme med HTTP, som er mer assosiert med forespørsel-svar type kommunikasjon, så måtte det gjøres med polling, som igjen fører med mer overhead enn med Websockets.[[1]](#footnote-1)

En annen fordel ved å velge Websockets som kommunikasjonsmetode var at Java støtter å sende og motta objekter over Websockets, som gjorde implementasjonen av koden betydelig simplere. I stedet for å ta imot en melding som en tekst-streng, JSON, XML eller annet og så parse det til et objekt, så kan jeg motta objektet direkte, og samtidig validere at dataen jeg mottar er på riktig format.

Beleilighet ble satt foran ytelse. Programmet sender ganske mye data med hver melding, og skulle det vise seg at systemet skulle sende millioner av meldinger til svært mange ressurser, slik at ytelse måtte bli høyere prioritert, ville jeg nok valgt en annen kommunikasjonsmetode. F.eks. kunne jeg ha brukt UDP, som funker godt når et kort svar skal sendes til en annen server kjapt uten noen form for handshakes, og som støtter «multicast», utsending til mange mottakere samtidig, noe som TCP (og dermed Webscokets) ikke støtter. Ved en implementasjon av UDP er det derimot enda viktigere å lage metoder som håndterer situasjoner hvor meldinger ikke kommer frem.[[2]](#footnote-2)

## Tilstander og tilstandslagring

Alle arver client

Egen klasse for meldinger

Egen klasse for tilstander

Enums for godkjente meldinger og tilstander

Tilstander

Legg inn flytdiagram for TM.

Alt lagres

Ukjent antall rm’s

Asynkron håndtering av meldinger

Synkron håndtering av oppgaver

Timeout fra TM, og ikke fra Rm

Beskrivelse av teknologi/design/arkitektur valg. Hva hadde jeg å velge mellom og hvorfor har jeg valgt det jeg har valgt.

# Fremtidig arbeid med oversikt over mangler og mulige forbedringer

Sikre kommunikasjonen. Burde gjøres.

# Eksempler som viser bruk av løsningen

# Installasjonsinstrukser

For å kunne kjøre programmet må Java SE 11 være installert. Java kan lastes ned fra <https://www.oracle.com/java/technologies/javase-downloads.html>

Klon prosjektet som ligger på linken <https://github.com/jobrajac/2phase-commit>.

Med git installert: git clone <https://github.com/jobrajac/2phase-commit.git>

Hvis du ikke har git installert kan du laste ned prosjektet som en ZIP.

## Kjøre prosjektet

Instruksjoner for å kjøre prosjektet med de eksemplene som er demonstrert i oppgaven:

Kjør klassen /2phase-commit/out/production/2phase-commit/Main.class ved hjelp av kommandoen:

java -cp . Main

Eventuelt kan du kompilere Java-filen /2phase-commit/src/Main.java selv, og så kjøre den kompilerte .class filen.

Loggfilene generert av TaskManager og alle ResourceManagers kan du finne i mappen /2phase-commit/logs/

# Lenke til kode-repository

<https://github.com/jobrajac/2phase-commit>

# Referanser

V. Raghavendra Prasad, S. M. (2010). *IMPLEMENTATION OF 2-PHASE COMMIT BY INTEGRATING THE DATABASE & THE FILE SYSTEM.* Angallu.

https://en.wikipedia.org/wiki/Two-phase\_commit\_protocol, mellom 01.04.2020 til 20.04.2020

https://docs.oracle.com/cd/B28359\_01/server.111/b28310/ds\_txns003.htm#ADMIN12222, mellom 01.04.2020 til 20.04.2020

1. https://en.wikipedia.org/wiki/WebSocket [↑](#footnote-ref-1)
2. https://en.wikipedia.org/wiki/User\_Datagram\_Protocol [↑](#footnote-ref-2)