Two Phase Commit

Av

Håkon Harnes

NTNU - Norges Teknisk-Naturvitenskapelige Universitet

Trondheim, 21 April 2020

Innholdsfortegnelse

Introduksjon

Dette er en rapport for en prosjektoppgave gitt i faget TDAT2004 - 'Datakommunikasjon med nettverksprogrammering'. Rapporten omfatter dokumentasjon av teknisk løsning samt begrunnelse for teknologi- arkitektur- og designvalg.

Prosjektet går ut på å implementere 2PC (Two Phase Commit) protokollen. Løsningen skal illustrere distribuerte transaksjoner som anvender 2PC. For å få fram alle aspekter ved protokollen skal noen av transaksjonene aborteres, mens andre skal committes.

Begreper

Programkoden er på engelsk, mens rapporten er på norsk. Det er derfor behov for følgende begrepsliste:

Engelsk	Norsk
Coordinator	Koordinator
Participant	Deltaker
State	Status
Abort	Aborter
Commit	Commit

Implementert funksjonalitet

Koordinator

- Kan starte en transaksjon
- Kan dele opp en transaksjon i del-transaksjoner
- Kan tildele hver del-transaksjon en deltaker
- Kan sende hver del-transaksjon til tilhørende deltaker
- Kan abortere en transaksjon
- Kan committe en transaksjon
- Kan sende abort-melding til alle deltakere i transaksjonen
- Kan sende commit-melding til alle deltakere i transaksjonen
- Kan sende melding til en deltaker
- Kan sende melding til alle deltakere som tilhører en viss transaksjon
- Kan motta statusendringer fra deltakere
- Kan motta stemmer (abort eller commit) fra deltakere
- Kan finne alle deltakere som er ledige
- Kan håndtere at deltakere kobler fra under transaksjon
- Kan skrive til loggen (write-ahead)
- Kan lese fra loggen
- Har timeout

Deltaker

- Kan motta en del-transaksjon fra koordinator
- Kan utføre en del-transaksjon
- Kan abortere en del-transaksjon
- Kan committe en del-transaksjon
- Kan sende stemme (abort eller commit) til koordinator
- Kan endre status
- Kan sende statusendring til koordinator
- Kan håndtere at koordinator kobler fra under transaksjon
- Kan skrive til loggen (write-ahead)
- Kan lese fra loggen
- Har timeout

Bruker

- Kan opprette konto
- Kan starte en transaksjon
- Kan stemme (abort eller commit) på en del-transaksjon
- Kan printe ID til deltaker
- Kan printe status til deltaker
- Kan printe liste med kommandoer

Brukeren oppnår dette ved å kjøre kommandoer i terminalvinduet. Disse kommandoene er beskrevet i delkapittel *Testing*.

Diskusjon

Denne delen av rapporten diskuterer og beskriver de ulike teknologi-, arkitektur- og designvalgene som ble gjort underveis.

Arkitektur- og designvalg

I 2PC kommuniserer koordinatoren med deltakerne over nett. De skal derfor ikke ha noen delte datastrukturer. Dette er grunnen til at det i løsningen er en pakke for koordinatoren, og en pakke for deltakeren. Ingen av klassene i disse pakkene er offentlige. De har altså kun pakketilgang, noe som sikrer skillet mellom koordinator og deltaker. En koordinator kan ikke bruke klasser fra deltaker-pakken, og vice versa. Det er heller ingen kommunikasjon mellom deltakere, alt går igjennom en koordinator.

Skillet mellom koordinatoren og deltakerne gjør at løsningen blir distribuert. Da kan systemet enkelt tilpasse seg ulike nettverksstørrelser. En annen fordel med denne arkitekturen er at koordinatoren er skjermet fra ond-artede transaksjoner. Dersom en transaksjon er ond-artet, vil dette kun påvirke de respektive deltakerene som kjører transaksjonen. Dette betyr at det er mer sannsynlig at en deltaker kræsjer framfor en koordinator, noe som er ønskelig. Hvis et kræsj eventuelt skulle oppstått kan koordinatoren fortsatt utføre nødvendige handlinger for å abortere transaksjonen. En ulempe ved distribuerte løsninger er økt ventetid på grunn av nettverksforespørsler.

Prosjektstrukturen er vist under. Legg spesielt merke til at `coordinator` og `participant` pakkene er adskilt fra hverandre. De deler på klassene `Account`, `LogManager`, `SubTransaction`, `Transaction` samt pakken `constants`. De deler derimot ikke på instanser av disse, slik at løsningen fortsatt er distribuert. I praksis kan disse flyttes inn i hhv. `coordinator` og `participant` pakkene for å få en helt distribuert prosjektstruktur, dog med mye duplikatkode.

```
- src/
    - logs/
    - twophasecommit/
        - constants/
            - Command
            - State
            - Vote
        - coordinator/
            - CommandHandler
            - CommandListener
            - Coordinator
            - Main
            - MessageHandler
            - MessageListener
        - participant/
            - CommandHandler
            - CommandListener
            - Participant
            - Main
            - MessageHandler
            - MessageListener
        - Account
        - LogManager
        - SubTransaction
        - Transaction
```

Både koordinatoren og deltakeren har hver sin `CommandHandler`, `CommandListener`, `MessageHandler` og `MessageListener`. Disse klassene lytter og prosesser meldinger fra hhv. brukeren og den respektive koordinatoren/deltakeren. Dette forenkler klassene `Coordinator` og `Participant` betraktelig. De inneholder kun hovedfunksjonaliteten til 2PC, mens de andre klassene tar seg av kommunikasjon mellom koordinator, deltaker og bruker. `Main` klassene oppretter nødvendige variabler, starter lyttere og starter eller kobler seg opp mot serveren.

For å forenkle løsningen er den ikke knyttet opp mot en database. Det er i steden for tatt i bruk klassen `Account` for å opprette testdata. Dermed får vi håndfast data i loggene uten å komplisere løsningen nevneverdig.

Hver koordinator og deltaker har sin egen undo/redo log. Den blir tatt i brukt ved abort eller feilhåndtering. Det er brukt Write-Ahead logging for alle transaksjoner.

Teknologivalg

Det var ingen krav til programmeringsspråk. Valget falt på Java, ettersom det er det vi har hatt mest undervisning i.

Det er to nettverksprotokoller som kan brukes til å overføre data mellom nodene: TCP og UDP. Begge protokollene har fordeler og ulemper. Disse er diskutert nedenfor.

2PC protokollen baserer seg på meldingsutveksling over nett mellom koordinatoren og deltakerne. Dersom en melding ikke kommer fram, vil transaksjonen aborteres etter en viss ventetid. Dette resultatet er uheldig, og noe vi ønsker å unngå for enhver pris. TCP er pålitelig og garanterer at meldingene kommer fram, mens UDP er upålitelig. Det åpenbare valget faller derfor på TCP. Dersom vi skulle brukt UDP måtte vi re-transmittert pakker etter en viss stund for å oppnå pålitelig overførsel. I praksis er det enklere å bruke TCP.

Det er likevel et par aspekter ved UDP som gjør at det er en fristende protokoll å anvende. Siden UDP er upålitelig, er den mer effektiv enn TCP. Da ville vi redusert ventetiden på nettverksforespørsler. Dette er, som nevnt i forrige delkapittel, en av ulempene ved distribuerte systemer. UDP støtter også broadcasting. Det vil si at man kan sende en melding til alle nodene i nettverket. Dette kan være nyttig for å spørre deltakerne om status, for eksempel for å finne ledige noder som kan utføre del-transaksjoner.

Det åpenbare valget her er TCP. Dette valget støttes også oppunder av Microsoft, som bruker 2PC over TCP i deres Host Integration Server. [1]

Fremtidig arbeid

Feilhåndtering

Det er allerede implementert en del feilhåndtering i løsningen. Blant annet vil en frakobling av koordinator eller deltaker føre til automatisk abort. Loggføringen er også korrekt.

Det som eventuelt må jobbes videre med er å lese loggen som er produsert, for så å avgjøre om man skal undo, redo eller undo-redo transaksjonen.

Automatisk avgjørelse

For å abortere eller committe en transaksjon må brukeren stemme. Dette er gjort for å se gangen i programmet. Et videre steg er å la programmet avgjøre om transaksjonen skal aborteres eller committes selv. Da kan den eksempelvis sjekke om ny saldo er negativ.

Grafisk brukergrensesnitt

Dette angår ikke implementasjonen av protokollen, men hvordan protokollen illustreres for brukeren. Løsningen bruker nå terminalen som grafisk brukergrensesnitt. En løsning med JavaFx eller Swing ville vært mer brukervennlig og oversiktlig.

Eksempler

Transaksjon som fullfører

Starter transaksjonen:

Koordinator

```
TRANSACTION A B 50
```

Første deltaker stemmer commit:

Deltaker 1

```
TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
YES
```

Andre deltaker stemmer commit:

Deltaker 2

```
TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
YES
```

Koordinatoren mottar commit-stemmene fra deltakerne, sender commit-melding og committer transaksjonen:

Koordinator

```
PARTICIPANT #60968: VOTED COMMIT TRANSACTION #1
PARTICIPANT #60972: VOTED COMMIT TRANSACTION #1

INITIATING GLOBAL COMMIT

LOG, WRITE: <1, COMMIT>

COMMITTING TRANSACTION #1...COMMITTED

LOG, WRITE: <1, END>
```

Deltakerne mottar commit-meldingen fra koordinatoren og committer:

```
Deltaker 1 / 2
DECISION: COMMIT

LOG, WRITE: <1_1, COMMIT>

STATE: COMMIT

COMMITTING SUB-TRANSACTION #1_1...COMMITTED

LOG, WRITE: <1_1, END>
```

De neste sidene viser hele terminalutklippene:

Koordinator

```
WAITING FOR PARTICIPANTS...

PARTICIPANT #60968: INITIALIZED
PARTICIPANT #60972: INITIALIZED
TRANSACTION A B

-- TRANSACTION 1 --
A -> B, 50.0

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #60968

-- SUB-TRANSACTION 1_2 --
B, 100.0 SUB(50.0), #60972

LOG, WRITE: <1, START>

SENDING SUB-TRANSACTIONS TO PARTICIPANTS...

LOG, WRITE: <1, A, B, 50.0>

PARTICIPANT #60968: PREPARING
PARTICIPANT #60972: PREPARING
PARTICIPANT #60968: VOTING
PARTICIPANT #60972: VOTING
PARTICIPANT #60968: WAITING
PARTICIPANT #60968: VOTED COMMIT TRANSACTION #1
PARTICIPANT #60972: WAITING
PARTICIPANT #60972: VOTED COMMIT TRANSACTION #1

INITIATING GLOBAL COMMIT

LOG, WRITE: <1, COMMIT>

COMMITTING TRANSACTION #1...COMMITTED

LOG, WRITE: <1, END>

PARTICIPANT #60968: COMMIT
PARTICIPANT #60972: COMMIT
PARTICIPANT #60968: INITIALIZED
PARTICIPANT #60972: INITIALIZED
```

Deltaker 1 / 2

```
STATE: INITIALIZED
CONNECTED TO COORDINATOR #1250
STATE: PREPARING

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #60968

LOG, WRITE: <1_1, START>

EXECUTED SUB-TRANSACTION #1_1

LOG, WRITE: <1_1, A, 100.0, 150.0>

STATE: VOTING

TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
YES

STATE: WAITING
VOTED: COMMIT
DECISION: COMMIT

LOG, WRITE: <1_1, COMMIT>

STATE: COMMIT

COMMITTING SUB-TRANSACTION #1_1...COMMITTED

LOG, WRITE: <1_1, END>

STATE: INITIALIZED
```

Transaksjon som ikke fullfører

Starter transaksjonen:

Koordinator

```
TRANSACTION A B 50
```

Første deltaker stemmer abort:

Deltaker 1

```
TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
NO
```

Koordinatoren mottar abort-stemmen fra deltaker 1, sender abort-melding og aborterer transaksjonen:

Koordinator

```
PARTICIPANT #61282: VOTED ABORT TRANSACTION #1

INITIATING GLOBAL ABORT

LOG, READ: <1, A, B, 50.0>

LOG, WRITE: <1, ABORT>

ABORTING TRANSACTION #1...ABORTED

LOG, WRITE: <1, END>
```

Deltakerne mottar abort-meldingen fra koordinatoren og aborterer:

```
Deltaker 1 / 2
DECISION: ABORT

LOG, READ: <1_1, A, 100.0, 150.0>

LOG, WRITE: <1_1, ABORT>

STATE: ABORT

ABORTING SUB-TRANSACTION #1_1...ABORTED

LOG, WRITE: <1_1, END>
```

De neste sidene viser hele terminalutklippene:

Koordinator

```
WAITING FOR PARTICIPANTS...

PARTICIPANT #61277: INITIALIZED
PARTICIPANT #61282: INITIALIZED
TRANSACTION A B 50

-- TRANSACTION 1 --
A -> B, 50.0

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #61282

-- SUB-TRANSACTION 1_2 --
B, 100.0 SUB(50.0), #61277

LOG, WRITE: <1, START>

SENDING SUB-TRANSACTIONS TO PARTICIPANTS...

LOG, WRITE: <1, A, B, 50.0>

PARTICIPANT #61282: PREPARING
PARTICIPANT #61277: PREPARING
PARTICIPANT #61282: VOTING
PARTICIPANT #61277: VOTING
PARTICIPANT #61282: WAITING
PARTICIPANT #61282: VOTED ABORT TRANSACTION #1

INITIATING GLOBAL ABORT

LOG, READ: <1, A, B, 50.0>

LOG, WRITE: <1, ABORT>

ABORTING TRANSACTION #1...ABORTED

LOG, WRITE: <1, END>

PARTICIPANT #61277: ABORT
PARTICIPANT #61282: ABORT
PARTICIPANT #61277: INITIALIZED
PARTICIPANT #61282: INITIALIZED
```

Deltaker 1 / 2

```
CONNECTED TO COORDINATOR #1250
STATE: INITIALIZED
STATE: PREPARING

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #61282

LOG, WRITE: <1_1, START>

EXECUTED SUB-TRANSACTION #1_1

LOG, WRITE: <1_1, A, 100.0, 150.0>

STATE: VOTING

TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
NO

STATE: WAITING    <-- KUN FOR DELTAKER 1
VOTED: ABORT      <-- KUN FOR DELTAKER 1
DECISION: ABORT

LOG, READ: <1_1, A, 100.0, 150.0>

LOG, WRITE: <1_1, ABORT>

STATE: ABORT

ABORTING SUB-TRANSACTION #1_1...ABORTED

LOG, WRITE: <1_1, END>

STATE: INITIALIZED
```

Transaksjon hvor deltaker ikke svarer

Starter transaksjonen:

```
Koordinator
TRANSACTION A B 50
```

Deltakerne blir bedt om å stemme:

```
Deltaker 1 / 2
TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
...
```

Koordinator venter i 15 sekunder uten å få svar. Aborter transaksjonen:

```
Koordinator
TIMEOUT - PARTICIPANTS TOOK TO LONG

INITIATING GLOBAL ABORT
```

Deltakerne mottar abort-meldingen fra koordinatoren og aborterer:

```
Deltaker 1 / 2
DECISION: ABORT

LOG, READ: <1_1, A, 100.0, 150.0>

LOG, WRITE: <1_1, ABORT>

STATE: ABORT

ABORTING SUB-TRANSACTION #1_1...ABORTED

LOG, WRITE: <1_1, END>
```

De neste sidene viser hele terminalutklippene:

Koordinator

```
WAITING FOR PARTICIPANTS...

PARTICIPANT #64775: INITIALIZED
PARTICIPANT #64780: INITIALIZED
TRANSACTION A B 50

-- TRANSACTION 1 --
A -> B, 50.0

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #64775

-- SUB-TRANSACTION 1_2 --
B, 100.0 SUB(50.0), #64780

LOG, WRITE: <1, START>

SENDING SUB-TRANSACTIONS TO PARTICIPANTS...

LOG, WRITE: <1, A, B, 50.0>

PARTICIPANT #64775: PREPARING
PARTICIPANT #64780: PREPARING
PARTICIPANT #64780: VOTING
PARTICIPANT #64775: VOTING

TIMEOUT - PARTICIPANTS TOOK TO LONG

INITIATING GLOBAL ABORT

LOG, READ: <1, A, B, 50.0>

LOG, WRITE: <1, ABORT>

ABORTING TRANSACTION #1...ABORTED

LOG, WRITE: <1, END>

PARTICIPANT #64775: ABORT
PARTICIPANT #64780: ABORT
PARTICIPANT #64775: INITIALIZED
PARTICIPANT #64780: INITIALIZED
```

Koordinator

```
STATE: INITIALIZED
STATE: PREPARING

-- SUB-TRANSACTION 1_2 --
B, 100.0 SUB(50.0), #64780

LOG, WRITE: <1_2, START>

EXECUTED SUB-TRANSACTION #1_2

LOG, WRITE: <1_2, B, 100.0, 50.0>

STATE: VOTING

TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
DECISION: ABORT


LOG, READ: <1_2, B, 100.0, 50.0>

LOG, WRITE: <1_2, ABORT>

STATE: ABORT

ABORTING SUB-TRANSACTION #1_2...ABORTED

LOG, WRITE: <1_2, END>

STATE: INITIALIZED
```

Transaksjon hvor koordinator kobler fra

Starter transaksjonen:

```
Koordinator
TRANSACTION A B 50
```

Deltakerne blir bedt om å stemme:

```
Deltaker 1 / 2
TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
...
```

Koordinator kobler fra. Deltakerne mister tilkoblingen:

```
Deltaker 1 / 2
STATE: DISCONNECTED
```

Deltakerne aborterer transaksjonen:

```
Deltaker 1 / 2

LOG, READ: <3_2, B, 100.0, 50.0>

LOG, WRITE: <3_2, ABORT>

STATE: ABORT
COULD NOT SEND STATE: ABORT

ABORTING SUB-TRANSACTION #3_2...ABORTED

LOG, WRITE: <3_2, END>

STATE: INITIALIZED
COULD NOT SEND STATE: INITIALIZED
```

De neste sidene viser hele terminalutklippene:

Koordinator

```
WAITING FOR PARTICIPANTS...

PARTICIPANT #64878: INITIALIZED
PARTICIPANT #64879: INITIALIZED
TRANSACTION A B 50

-- TRANSACTION 1 --
A -> B, 50.0

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #64878

-- SUB-TRANSACTION 1_2 --
B, 100.0 SUB(50.0), #64879

LOG, WRITE: <1, START>

SENDING SUB-TRANSACTIONS TO PARTICIPANTS...

LOG, WRITE: <1, A, B, 50.0>

PARTICIPANT #64878: PREPARING
PARTICIPANT #64879: PREPARING
PARTICIPANT #64878: VOTING
PARTICIPANT #64879: VOTING
^C%
```

Deltaker 1 / 2

```
STATE: INITIALIZED
CONNECTED TO COORDINATOR #1250
STATE: PREPARING

-- SUB-TRANSACTION 1_1 --
A, 100.0 ADD(50.0), #64878

LOG, WRITE: <1_1, START>

EXECUTED SUB-TRANSACTION #1_1

LOG, WRITE: <1_1, A, 100.0, 150.0>

STATE: VOTING

TYPE 'Y/YES' FOR COMMIT OR 'N/NO' FOR ABORT
STATE: DISCONNECTED

LOG, READ: <1_1, A, 100.0, 150.0>

LOG, WRITE: <1_1, ABORT>

STATE: ABORT
COULD NOT SEND STATE: ABORT

ABORTING SUB-TRANSACTION #1_1...ABORTED

LOG, WRITE: <1_1, END>

STATE: INITIALIZED
COULD NOT SEND STATE: INITIALIZED
```

Merk at deltakeren ikke får sendt status til koordinatoren. Dette har ingenting å si, da koordinatoren er frakoblet. Det viktigste er at hver del-transaksjon blir abortert.

Installasjonsinstruksjoner

Last ned programkoden fra blackboard eller clone git repositoriet med:

```
git clone https://gitlab.stud.idi.ntnu.no/haakaha/2pc.git
```

Naviger til `2PC/src`. Koden kan deretter kompileres med:

```
javac twophasecommit/coordinator/Main.java
javac twophasecommit/participant/Main.java
```

Testing

Koordinatoren startes med:

Koordinator

```
java twophasecommit/coordinator/Main
```

Da vil koordinatoren vente på at deltakere kobler seg til. Deltakerne startes med:

Deltaker

```
java twophasecommit/participant/Main
```

Deltakeren vil da være koblet til koordinatoren. For at en transaksjon skal kunne kjøres, må minimum to deltakere være koblet til.

Videre kan man starte en transaksjon med:

Koordinator

```
TRANSACTION A B 50
```

Deretter vil deltakerne bli bedt om å stemme på om transaksjonen skal committes eller aborteres. Legg inn commit-svar med:

Deltaker

```
YES
```

eller abort-svar med:

Deltaker

```
NO
```

Da har transaksjonen enten blitt committet eller abortert. Status for deltakerne er nå `State.INITIALIZED` og vi kan kjøre en ny transaksjon dersom det er ønskelig.

Det er også mulighet for å definere egne kontoer. Koden under lager to kontoer, sparekonto og BSU. De har saldo på hhv. 100 og 50.

Koordinator

```
ACCOUNT SPAREKONTO 100
ACCOUNT 'SPAREKONTO' ADDED

ACCOUNT BSU 50
ACCOUNT 'BSU' ADDED
```

For å utføre en transaksjon på disse kontoene kan vi skrive:

Koordinator

```
% TRANSACTION SPAREKONTO BSU 50
```

Under er en oversikt over kommandoer som brukes for å styre programmet. Det er ikke alle kommandoene som kan brukes i deltaker-terminalen. Skriv 'HELP' for en liste med kommandoer.

Beskrivelse	Kommando
Opprett konto	`ACCOUNT <NAME> <BALANCE>`
Start transaksjon	`TRANSACTION <ACC1> <ACC2> <AMOUNT>`
Print deltakerstatus	`STATE <participant id>`
Print egen status	`STATE`
Stem abort	`Y/YES`
Stem commit	`N/NO`
Print egen id	`ID`
Print kommandoer	`HELP`

Dersom vi kjører:

```
Koordinator
TRANSACTION A B 50
```

Vil programmet starte en transaksjon med konto A og B. Hvis kontoene ikke finnes, vil programmet lage disse med saldo lik 100.

Referanser

[1] Microsoft. (2017, nov) How to perform a two-phase commit transaction over tcp/ip. (Lasted ned 17.04.2020). [Online]. Available: https://docs.microsoft.com/en-us/host-integration-server/core/how-to-perform-a-two-phase-commit-transaction-over-tcp-ip2