TDAT2004 - Prosjekt: 2-PC-Protocol

Gultvedt, Even
Hollum, Jørgen
Younger, Eric

21.04.2020

GitHub-lenke: https://github.com/ericyounger/2PC-protocol

Innhold

1 Introduksjon

I faget TDAT2004 "Datakommunikasjon med nettverksprogrammering" har vi fått oppgaven å implementere en "to-fase-commit" løsning. Det kan leses mer om hva dette er på denne lenken. Et viktig punkt å notere seg for oppgaven er at all kode er produsert av utviklerne i teamet, bestående av artikkelforfatterne. Videre dokumentasjon enn det som finnes i denne artikkelen kan finnes i siste kapittel: 8.

2 Implementert funksjonalitet

Vi valgte å gå for C++ som språk for vår løsning, da dette er et raskt språk, og vi ønsket å lære å programmere i dette bedre. Mer om grunner for valg av teknologi, arkitektur og design i kapittel 3. I vår interpretasjon av oppgaven har vi sett mange likheter til git, og vi kommer til å bruke disse likhetene til å forklare mange av våre funksjoner.

2.1 Klient

En bruker møter et enkelt tekst-brukergrensesnitt, hvor han kan velge en mappe som skal overvåkes av vår løsning. Dette blir noe av det samme som *gitinit*. Deretter kan brukeren "committe" filene sine. Vi har ikke et mellomledd som skiller mellom det git caller "commit" og "push", men kun noe som likner på "git push".

Om brukeren velger å committe sine filer, blir det sendt en endringslogg til server med brukerens endringer siden siste suksessfulle commit. Den får så tilbake enten en beskjed om at committ-kommandoen var vellykket, eller at systemet har en konflikt med filer som er på de andre brukernes maskiner. Om en slik feil oppstår, vil brukeren få beskjed om hvilke filer det gjelder, og hva som er forskjellen mellom de to filene. Da må brukerne snakke sammen og finne ut av hvordan de skal fikse denne feilen. Vi har også muligheten til å sette brukerens fil tilbake til det som befinner seg på server, en slags "rollback". Etter en suksessfull commit fjernes alt innhold fra endringsloggen.

2.2 Server

Serveren settes raskt opp og venter på tilkoblinger fra klienter. Hver klients commit-spørring blir plassert i en kø og tas hånd om en og en. Dette vil kun føre til flaskehalser om teamene som arbeider mot serveren er veldig store, og selv da er det nødvendig å beholde midtpunktet i to-fase-commit, transaksjonen.

Når klientens spørring er i gang, sender den over sin lokale endringslogg. Server sender så denne til alle andre tilkoblede klienter, som sjekker om det er noen konflikter mellom denne og deres endringslogg, og sender resultatet tilbake til server. Det er en tråd per tilkobling for å minske ventetiden mellom spørring fra server og resultatet fra alle klientene.

Serveren sender alle eventuelle feilmeldinger tilbake til den originale klienten, eller en beskjed om at commit-en er vellykket. Om det er tilfelle, mottar den alle de endrede filene fra den originale klienten, og oppdaterer sine filer på server. Til slutt oppdateres endringsloggen på server, og klienten får beskjed om at commit-en er ferdig.

2.3 Klasse og metode forklaringer

2.3.1 Klient

- TUI er klientens interface. TUI står for "Text User Interface" og gir brukeren valg i fhr. til to-fase-commit løsningen. TUI spør etter prosjektets filsti, og initialiserer da en "ChangedFiles.txt" i prosjektet som den skriver endringslogg til. Når bruker tar valg som "Commit" sendes "ChangedFiles.txt" over TCP forbindelsen med server, og den blir sendt videre til andre tilkoblede klienter for å sjekke om commiten går igjennom ved en avstemming.

- Klassen "File" lagrer informasjon om filer; filsti, filnavn, sist modifisert-dato og innhold i fil. Den har flere GET og SET metoder, og flere konstruktører slik at man kan lage objekter med ulikt innhold.

- FileComparator er en sammenligningsklasse med kun statiske metoder som tar inn to objekter av typen "File" og sammenligner de. De inneholder to statiske metoder: "fileDiff" og "newestFile".

  - Metoden fileDiff returnerer en boolean som er true hvis filene er forskjellige og ellers false.
  - Metoden newestFile returnerer en integer mellom [-1, 1] som indikerer hvilken fil som er nyest; Det returneres 1 hvis første parameter er nyest, 0 hvis de er like gamle, og -1 hvis første parameter er eldst. Se File klassen for mer informasjon.

- ChangelogParser er en klasse med to statiske metoder. Metodene brukes for å skille en lang streng og dele informasjonen opp i "tokens" som kan bli konvertert til File objekter, som så blir satt i en liste og returnert.

  - Metoden parseToFiles har 3 parametre, filsti, en skiller(delimiter) som skiller på filer, og en skiller(delimiter) som skiller på variabler. Denne metoden bruker vi for å hente ut informasjon fra en fil "ChangedFiles.txt" som oppdateres gjennom en tråd som sjekker etter endrede filer siden sist commit.
  - Metoden parseToFilesFromString tar inn en streng, en skiller(delimiter) som skiller på filer, og en skiller(delimiter) som skiller på variabler. Denne metoden bruker vi for å hente ut informasjon fra en streng.

- SocketClient er en klasse som står for TCP kommuniksjonen fra klienter til server. Klienter bruker denne til å koble seg opp til Serveren. Klassens konstruktør tar inn ip adressen til hvor serveren befinner seg og hvilken port serveren lytter på.

  - Metoden sendCommit er en privat metode i klassen. Denne sender .changelog til servereren gitt en socket-descriptor (linux-spesifikk fil) og filepathen til .changelog filen.

- commit er en public void metode som tar i bruk den private metoden sendCommit. Filepathen til .changelog sendes inn som parameter og brukes videre som parameter i sendCommit for å kunne sende en commit.

2.3.2 Server

- ServerSocket er en klasse som setter opp en TCP forbindelse som server. Server står og lytter etter klienter som kobler seg til. Serveren fungerer som Koordinator i to-fase-commit protokollen, hvor den benytter seg av worker-threads for å effektivisere kommunikasjonen mellom server og klient for lesing, skriving, og stemmehåndtering. I tillegg har den en egen worker thread til eventloopen. Denne tar seg av hovedflyten i programmet. Vi kan for eksmpel kun håndtere en commit-spørring av gangen, da dette er transaksjonene i systemet.

  - Metoden writeChange tar endringer fra klienter og legger de til den globale loggen over endringer fra server.
  - Metoden commitRequest tar for seg å gi beskjed til alle klienter om å stemme for om en commit går igjennom eller ikke.
    Metoden håndterer også stemmene i etterkant og gir beskjed til klientene tilkoblet om committen gikk igjennom eller ikke.
  - Metoden updateServerFiles mottar alle de endrede filene over TCP-forbindelsen fra klienten, og lagrer disse i passende mapper på server.

2.3.3 Klasser felles for klient og server

- Worker er en klasse som initialiser et bestemt antall med tråder som som står klare til å ta arbeid som kommer inn. Vi bruker Worker klassen til å lage "Event loop" for hovedkontroll flyt og bruker det også for å lage såkalte "Worker-threads", som får arbeid delegert ifra hovedkontroll flyten("Event loop"). Koden for Worker hadde vi skrevet allerede til Øving 2 i TDAT2004, og er endret smått for å tilpasses til vårt bruk.

- FileChecker er en klasse med kun statiske metoder. Metodene i klassen er for skriving og lesing til filer. Selve klassen har som hovedfunksjon å behandle data til og fra filen ChangedFiles.txt. Enkelte metoder returnerer en integer eller vector med informasjon.

  ChangedFiles.txt inneholder et "nulltidspunkt", informasjon om filnavn og sist endret dato for filer som er endret sammenlignet med "nulltidspunktet". Med "nulltidspunkt" menes at ved oprettelse- eller oppdatering av et prosjekt må det settes en tid, "nulltidspunkt", for å kunne sammenligne sist en fil er endret siden opprettelsen eller oppdateringen. Dersom filens endringstid er nyere enn "nulltidspunktet" vet vi at denne filen er blitt endret på. Filer som er blitt endres legges da i filen ChangedFiles.txt

- Metoden createLastModifiedFile oppretter ett "nulltidspunkt" og skriver det til filen "ChangedFiles.txt".
- Metoden getTimeOfProjectUpdate henter ut "nulltidspunktet" fra ChangedFiles.txt.
- Metoden writeAllModifiedFilesToFile sjekker om filer er blitt endret siden "nulltidspunktet" og skriver disse til filen "ChangedFiles.txt"
- Metoden getAllFilesInProject henter ut alle filer som finnes i prosjektet og returner en liste(vector) med File objekter som lagrer informasjon om filene.
- Metoden getFilesChanged returner en integer med antall av filer som har blitt endret.
- Metoden split tar inn en streng og en splitter(delimiter) og returnerer en liste(vector) med de oppdelte strengene.
- Metoden getAllChangedFiles henter ut alle modifiserte filer fra "ChangedFiles.txt" og returner en liste(vector) av alle filnavn som er endret i form av strenger.
- Metoden getFileContent henter ut innholdet fra en fil gitt filstien til filen ifra parameter til metoden.
- Metoden getChangedFiles henter ut alle endret filer fra "ChangedFiles.txt" og returner en liste(vector) med File objekter som inneholder informasjon om filene bestående av filnavn, innhold og filepath. Se File klassen for mer informasjon om lagring av informasjon.

3 Diskusjon av teknologi-, arkitektur og designvalg

Som nevnt tidligere har vi valgt å implementere vår løsning i C++. På dette punktet hadde vi mange valg, blant andre Java, Go og Rust. Vi valgte C++ både fordi dette er et raskt språk, fortsatt mye i bruk, og fordi vi ønsket å lære mer om det. Vi kunne også litt grunnleggende, siden vi hadde gjort fagets tidligere øvinger i dette språket.

2-PC-protokollen er basert på en transaksjon mellom et sett deltagere (klienter) og en koordinater (server). Denne transaksjonen kan bokføres på forskjellige måter, men vi valgte å ha en fil på server som holder styr på hvilke handlinger som utføres hos deltakerne. Et alternativ til dette kunne være å ha en databasetilkobling med loggføringen, men dette anså vi som for tungvint for løsningen, og ville legge til et ekstra ledd med nettverksoverføring av informasjon som strengt tatt ikke var nødvendig for vår løsning. C++-kode har også små forskjeller basert på hvilket operativsystem en bruker det på, så vi valgte å bruke linux-kommandoer for ting som socket-tilkoblinger. Siden serveren uansett hadde vært kjørt på en Linux-maskin gjennom continous deployment på NTNUs nett virket dette som den naturlige løsningen. Alle team-medlemmene har UNIX-maskiner (Linux eller Mac), som også var en god grunn til å holde oss til denne varianten av språket, og ikke bruke Microsofts C++-variant.

Selve løsningens arkitektur er basert på socket-tilkoblinger mellom en server og et sett klienter. Vi har et tekstbasert brukergrensesnitt som brukerne samhandler med. Kommandoer går til en instans av en socket-klient klasse, som kobler seg til server. Her går instruksjoner og data frem og tilbake over en TCP-tilkobling i form av tekst. Vi sender over en TCP-forbindelse fordi dette er en pålitelig overføringsmetode. Å velge UDP i denne sammenheng ville ikke gitt mening, da vi er avhengig av at alle instruksjoner og all data kommer frem og blir mottatt på riktig måte. TCP-socketen tar seg av den underliggende kommunikasjonen, og vi kan fokusere på å sende informasjon. Grunnen til at vi valgte å kun sende tekstlig informasjon er for enkelhetens skyld. Vi kunne også serialisert og deserialisert data, men vi ønsket å fokusere vårt arbeid på selve protokollen vi skulle implementere.

Videre så har vi benyttet oss av enkelte "design patterns" som skulle støtte opp mot arkitekturen til 2-PC-protokollen. Av de vi kan nevne har vi benyttet oss av "Seperation of Concerns"(SOC), ved å dele opp klassene i forskjellige filer. Det er også laget en del statiske metoder som kan kalles uten å måtte lage instanser av klassen, dette gjør igjen at koden blir ryddigere, og er gjenbrukbar. Design patternet "Worker-threads" også kalt "Thread pool" er benyttet siden koden er naturlig hendelsesdreven(Event-Driven). Worker klassen benytter seg av "Concurrency Patterns" som "Lock" og "Monitor". Lock mønsteret benytter seg av en lås som hindrer at andre gjør endringer på samme filer, med dette så

unngår vi "Race Conditions". For å ikke sløse CPU-kraft så benyttes det også "Monitor" med "Condition Variables". Da blir trådene satt på vent, mens de venter på å bli notifisert når det er arbeid for trådene å gjøre. Da unngår vi kjente problemer med "spin-locks" som konkurrerer over låsen og låser som går på tomgang(Busy-Waiting).

Transaksjonen i 2-PC-protokollen er kjernen i løsningen vi har produsert, og denne har vi løst ved å lagre lokale endringer i en tekstfil. Det går en tråd i bakgrunnen på klienten som sjekker etter oppdateringer på filer, og legger til hvilke filer som ble endret når. Tråden stopper når klienten ønsker å committe data, og endringssjekken kjører en siste gang. Denne løsningen er ikke særlig elegant, men utfører sitt oppdrag. Et alternativ kunne vært å delt opp prosessen slik git gjør, i commit og push. Vi ønsker derimot å gjøre prosessen enklest mulig for bruker, og utfører derfor denne prosessen automatisk. Dette er lettere for oss, da vi ikke har lagt inn muligheten for det git kaller branching.

4 Fremtidig arbeid og videreutvikling

Vår implementasjon av to-fase-commit er svært enkel i forhold til store versjonskontroll-systemer som for eksempel git, og derfor har vi kommet opp med en liste med forbedringer som kunne vært gjort for å øke brukervennligheten og effektiviteten til vår løsning. Vi ønsker også å vise til at dette er en minimal Minimum Viable Product-løsning, og har fortsatt en del bugs vi gjerne skulle hatt tid til å utradere før levering.

4.1 Klient

Som nevnt i diskusjonskapittelet over har vi et ønske om å implementere serialisering og deserialisering av info som beveger seg over TCP-forbindelsen, og ikke sende kun tekst. Dette hadde gitt oss muligheten til å sende andre typer filer enn kun tekstfiler, og det hadde også vært enklere å implementere delløsninger som komprimering og kryptering av innholdet som sendes over forbindelsen.

Muligheten for lokal og sentral branching av prosjektene hadde også vært et stort steg videre i utviklingen av løsningen om dette skulle vært tatt i bruk i praksis. Branching hadde gitt deltakerne i prosjektet større mulighet til å jobbe i team og holde sin kode oppdatert på server uten nødvendigvis å holde alle deltakeres klienter synkronisert til enhver tid. En slik løsning kunne vært implementert ved hjelp av en ID for branchene for både commit og pull, og at filsjekker kun skjedde på relevant branch. Vi har jobbet en del med en tilleggsfunksjonalitet, hvor bruker kan "pulle" filene som ligger på server og oppdatere sine lokale filer med andres oppdateringer. "Pulling" ble desverre ikke ferdigstillt til prosjekt innlevering, men det er noe vi ønsker at skulle blitt ferdig implementert i prosjektet.

4.2 Server

Et stort punkt som hadde økt skalerbarheten og brukervennligheten til serveren hadde vært å hatt flere servernoder. Dette kunne vært implementert ved at en commit brukte dybde-først-søk gjennom alle tilkoblede server-noder og satte i gang en commit-spørring derfra. Deretter kunne resultater fulgt de samme veiene tilbake, med en liste med feil og brukere de kom fra. Dette ville senket behovet for en stor fysisk server som kunne fått en lang kø av commit-spørringer om teamene skulle vokse seg store.

En annen videreutvikling for serveren som er litt mindre i omfang hadde vært å sette opp continous deployment. Å hoste serveren på NTNUs nett og kjøre oppdateringer ved push til master-grenen på et gitlab-repo for prosjektet hadde økt brukervennligheten i oppstartsfasen, for da hadde det ikke vært nødvendig å hoste det selv.

Muligheten for flere prosjekter hadde selvfølgelig også vært en god forbedring

for serveren. Dette kunne vært løst ved at hver tilkoblede klient sendte med en teamID, eller ble tildelt en som kunne vært delt videre. Da hadde implementasjonen av en enkel fler-prosjekt-løsning blitt en realitet uten mye vanskeligheter.

5 Eksempler på bruk

2CP-løsningen vi har laget kan brukes på forskjellige områder. Som beskrevet tidligere i rapporten har vi tatt stor inspirasjon fra ulike git løsniger. Denne 2CP-løsningen kan derfor være en "erstatning" for løsninger gitt av tjenester som f.eks Github og GitLab. Løsningen vår er kanskje ikke konkurrerbar med disse, men grunn prinsippene er like. Personer som vil sammarbeide på et prosjekt for å utvikle et produkt ved hjelp av koding, har nå muligheten til å jobbe sammen med delt kode som til en viss grad ikke skaper konflikter.

Løsningen kan også fungere som en google drive da flere deltakere kan holde på felles lagring av data. Filer kan legges inn og bli delt med andre - til en viss grad, og personer kan skrive på samme tekstdokument.

2CP-løsningen kan være med på å forhindre phantom-lås og andre utfordringer ved transaksjoner, samtidig som å gi en konfliktfri indikasjon på overføring av data mellom flere parter.

Endelige resultater ifra bruk:

```
Client 1:\\
/home/evengul/CLionProjects/2PC-protocol/prosjekt/cmake-build-debug/client\\
Client is running
Connected with status 0

Enter project filepath:
/home/evengul/CLionProjects/2PC-protocol/prosjekt/
The filepath is valid
Starting up..
Ready

What would you like to do?
1. Commit  | 2. Pull  | 3. Exit
1
You chose: 1

Sending commit message, awaiting response..
Sending changelog to server
Data sent to server for commitment
Commit is accepted
28 files are being sent to server
The upload was successful, keep on working.


What would you like to do?
1. Commit  | 2. Pull  | 3. Exit
1
```

```
You chose: 1

Sending commit message, awaiting response..
Sending changelog to server
Data sent to server for commitment
We have a merge error that needs resolution before commit can take place

Client 2:
Enter project filepath:
/home/evengul/CLionProjects/2PC-protocol/prosjekt/
The filepath is valid
Starting up..
Ready

What would you like to do?
1. Commit  | 2. Pull  | 3. Exit
1
You chose: 1

Sending commit message, awaiting response..
Sending changelog to server
Data sent to server for commitment
Commit is accepted
1 files are being sent to server

Server:
/home/evengul/CLionProjects/2PC-protocol/prosjekt/cmake-build-debug/server
Server is running. Waiting for clients...
New client has connected
Waiting for clients...
No other clients connected. Instant accept
Client has been told of this.
Updating server files

28 files
Ready to take in files
Successful in this
Commit is complete
New client has connected
Waiting for clients...
Asking clients for commit votes
Updating server files

Unsuccessful in this
```

6 Installasjonsinstruksjoner

Kjøringen av programmet er relativt enkel, en laster ned repoet, og har kjøringsfilene direkte tilgjengelig.

Installasjon

```
git clone https://github.com/ericyounger/2PC-protocol
cd 2PC-protocol/prosjekt
./server # I en terminal
./client # I en annen terminal
```

Om en ønsker å kompilere kildekoden selv, er det følgende kommandoer som er brukt:
Server: $g++\ main.cpp\ -std=c++17\ -o\ server\ -pthread$
Klient: $g++\ TUI.cpp\ -std=c++17\ -o\ client\ -pthread$

7 Hvordan teste løsningen?

Vi har satt opp en egen main metode i filen "tester.cpp" for automatisk testing av metoder tatt i bruk i 2CP-protokollen. Dette ved hjelp av oppkobling mot Github Actions. For hver gang vi pusher til master- eller testing branchen vil denne filen kompileres og kjøres. Dette gjør at vi får automatisk testet enhetstestene. Vi kan da holde en oversikt over antall feil og hvilke metoder som ikke fungerer slik som de skal. Eksempel på en vellykket test til prosjeketet kan sees her: https://github.com/ericyounger/2PC-protocol/runs/606540638 For videre lesing om hvordan dette er gjort åpne Workflows/ccpp.yml filen i prosjektet på Github.

I og med at vi ikke fikk lov til å benytte tredjepartsbiblotek i oppgaven, så var dette vår løsning på enhetstesting. Integrasjonstesting er gjort manuelt med flere utskriftslogger(cout) opp igjennom hele prosjektet for å se at ønsket resultat forekommer. Disse utskriftsloggene er stort sett fjernet og/eller kommentert ut ved innlevering av prosjekt.

Siden serveren ikke er publisert på noen server, så kreves det at man først starter serveren lokalt på egen maskin. Den kan startes med å kjøre kommandoen "./server" i fra prosjekt mappa. Deretter starter man klienten med kommandoen ".client". år klientenprogrammet har blitt startet, får bruker opp en "TUI"(Text-User-Interface) hvor bruker blir spurt om en filsti til roten av prosjektet. Dersom stien ikke finnes så får bruker valget om å opprette stien og benytte seg av den stien. Når brukeren har tastet inn en gyldig filsti så kjører klientprogrammet. Da har brukeren valgene:

- 1. Commit
- 2. Pull (Ikke implementert enda)
- 3. Exit

8 API-dokumentasjon

Her er en lenke til vår API-dokumentasjon