Primary Replication Framework

Andreas Heim (andrheim)
INF5510 Hjemmeeksamen 2 • 25 Mai 2010

Introduksjon og forutsetninger

Rammeverket for primary copy replication er utført i henhold til oppgaveteksten. Det har dessverre ikke vært tid til å implementere test-eksemplene gitt i oppgaven, men der er implementert et enklere testeksempel som viser hvordan de replikerte objektene interagerer med primær-kopi og rammeverket forøvrig. Gjennomgående i applikasjonen brukes nodens LNN som unik identifikator, og applikasjonen vil ikke oppføre seg korrekt hvis LNN ikke er unikt.

Ordliste

I dette dokumentet kommer følgende bekgrep til å bli brukt gjennomgående:

Framework: Hele den funksjonelle delen av rammeverket. Er også klassenavn for rammeverket.

Replicable: Objektet/klassen som kan replikeres i rammeverket.

Primary Framework: Den delen av rammeverket som vedlikeholder replikaer, og primær kopi.

Worker: Den delen av rammeverket som sørger for at Primary Framework er tilgjengelig, og utnevner et nytt hvis Primary Framework er nede.

Funksjonell beskrivelse

Replikering

Når replikering blir initiert via et kall på replicateMe-metoden vil det bli klonet og kopiert ut et antall replikaer på forskjellige noder. Nodene blir villkårlig valgt. Hvis antall replikaer som skal vedlikeholdes er større enn antall ledige noder, så blir alle ledige noder fylt opp.

Det er logikk for å ikke plassere to replikaer på samme node, og heller ikke på samme node som primærkopi.

Ved replikering av objekter blir det også plassert ut en Worker på hver node det eksisterer en replika på. Hver worker får en egen id.

Replikaer og Workere blir registrert i hver sin Map, indeksert på den aktuelle nodens LNN (logical node number). Replika og worker blir også fix'et på noden.

Håndtering av endringer på primærkopi og replika

Objekter av type Replicable har en metode addObserver[Framework] som må sette en lokal variabel med Primary Framework. Når et Replicable-objekt endres skal det kalle metoden notify[Replicable] på Framework med seg selv som parameter.
Rammeverket vil så avgjøre om man snakker om den primære kopi eller en replika.
Hvis det er primærkopi som er endret vil rammeverket iterere over alle replikaer og kalle metoden primaryCopyUpdated[Replicable] på hver replika, med primærkopi som argument. Det er da opp til objektene som implementerer typen Replicable å avgjøre hva de vil gjøre med dette objektet. I testeksempelet vedlagt vil replikaene hente ut all informasjon fra objektet og oppdatere sin egen informasjon.

Hvis en replika er oppdatert vil rammeverket bare informere primærkopi ved å kalle metoden replicaUpdated[Replicable] på primærkopi. Primærkopi kan da velge å oppdater sin egen informasjon, og har da ansvar for å si ifra til rammeverket at det har oppdatert informasjonen sin.

Vedlikehold av replikaer

Hvert annet sekund sjekker Primary Framework at alle replikaer samt primærkopi fortsatt er tilgjengelige. Dette gjøres ved hjelp av følgende enkle teknikk.
Det itereres over alle replikaer, replikaens nodes LNN blir lagt til i en array (inactiveList).
Deretter kalles det en locate på replikaen, på neste linje fjernes LNN fra inactiveList.
Hvis locate feiler vil ikke LNN fjernes fra arrayen, og vil bli slettet når prosessen starter igjen etter to sekunder. Workeren på samme node vil også bli slettet.
Dette er en meget enkel og grei teknikk, men har den begrensingen at rammeverket ikke kan oppdage mer enn en sviktet replika pr annet sekund.

Når en replika forsvinner vil rammeverket forsøke å lage en ny replika på en ny node, gitt at det er flere noder tilgjengelige. Det lages også en ny Worker på samme ledige node.

Vedlikehold av primærkopi

Primærkopi vedlikeholdes på samme måte som en replika. Forskjellen er at LNN blir ikke lagt til i inactiveList, isteden legger man til "-1" for lettere å kunne identifisere at det er primærkopi som har forsvunnet.
Det blir så satt igang en prosess for å forfremme ny primærkopi. Her velges alltid den første replikaen som ligger i listen, pga. sjansen for at denne inneholder den nyeste informasjonen er størst.

Vedlikehold av replikeringrammeverket

På hver node med en replika ligger det også en Worker. Workeren er et objekt av samme klasse som Primary Framework, men med isPrimaryFramework satt til false. Dette gir da en annen oppførsel enn et Primary Framework. Hvert annet sekund vil den hente informasjon fra Primary Framework. Informasjonen består av: Primær kopi, alle replikaer, alle workers, siste worker-id og hvor mange kopier som skal vedlikeholdes.
Denne informasjonen vil da ligge klar hvis Primary Framework går ned.

I det tilfellet hvor Primary Framework er utilgjengelig vil alle Workers igangsette en prosess for å velge et nytt Primary Framework. I denne perioden er systemet sårbart, iom at endringer ikke vil propagere til replikaer.

Utvelgelsen gjøres på meget enkel basis. Hver av Workerene sover i 4 sekunder x sin egen unike workerId.

Utfordringer

Threadsafing

Jeg fikk en del uventede problemer med Map-klassen, og endte opp med å thread-safe denne vha monitor og Conditions.

Utfordringer med Emerald

I min opprinnelige løsning var det lagt opp til at man kunne instansiere f.eks. et Framework.of[ValueObject] og få objekter med riktig type tilbake ved kall på getPrimaryCopy etc.

Her fikk jeg problemer på grunn av at Framework.of[ValueObject] må inneholde en liste med andre Framework.of[ValueObject] som workere. Dette ga en out of memory-error på compiler. Jeg prøvde også å lage en egen Worker-klasse som kunne parameteriseres med type. Men denne måtte igjen inneholde et Framework.of[ValueObject], også dette ga oom-error ved typesjekking. Det gikk derfor med litt tid til feilsøking og å programmere en ny løsning.

Mulige feilscenarier

Utvelgelse av nytt Primary Framework

Ved meget høyt antall noder vil de nodene med høy id sove unødvendig lenge, og ikke kunne holde rede på Primary Framework.

Hvis Primary Framework går ned før alle nodene har mottatt lister med alle Workers vil man også kunne komme i en situasjon der man får to Primary Frameworks. Dette er løst ved at idet man setter Primary Framework (metoden setPrimaryFramework[Framework]) så settes også primaryFramework til "false". Dermed vil det være en mulighet for at en Worker som setter igang utvelgelse av primary framework gjør om et tidligere Primary Framework til en Worker. Dette er på ingen måte en failsafe, men løser de fleste tilfeller.

Hvis nettverket blir segmentert finnes det heller ingen mekanisme for å flette sammen informasjon fra to Primary Frameworks, da disse ikke vil ha kjennskap til hverandre.

Identifisering av primærkopier

Om en Replicable er primærkopi eller en replika avgjørdes vha. metodene getIsPrimaryCopy og setIsPrimaryCopy på Replicable. Dette er for gjøre notfy til Framework så enkelt som mulig. Ulempen med dette er selvfølgelig at andre prosesser enn de som tilhører Frameworket kan sette denne verdien og således føre til korrumpering av tilstanden. Dette finnes det ingen sjekker mot.

Forbedringspunkter

Egen Worker-klasse

For å rydde opp i koden og øke lesbarheten kunne det vært fornuftig å innføre en egen worker-klasse med logikken og oppførselen som hører til Worker-delen.

Map-klassen

Et stykke ut i utviklingen så jeg at mye av funksjonaliteten for å fjerne inaktive noder kunne blitt flyttet ut i Map-klassen. Denne ville da hatt en egen prosess som fjernet objekter som ikke kunne lokaliseres.

Testeksempel

Rammeverket er testet med et meget enkelt objekt, Complex, som inneholder to variable, Imag og Real.

Objektet implementerer metodene rammeverket krever i henhold til beskrivelsene i denne rapporten.

Eksempelet kjøres fra testexample.m og gjør følgende:

Lager et objekt av type Complex.

Replikerer det på to noder.

Venter fem sekunder.

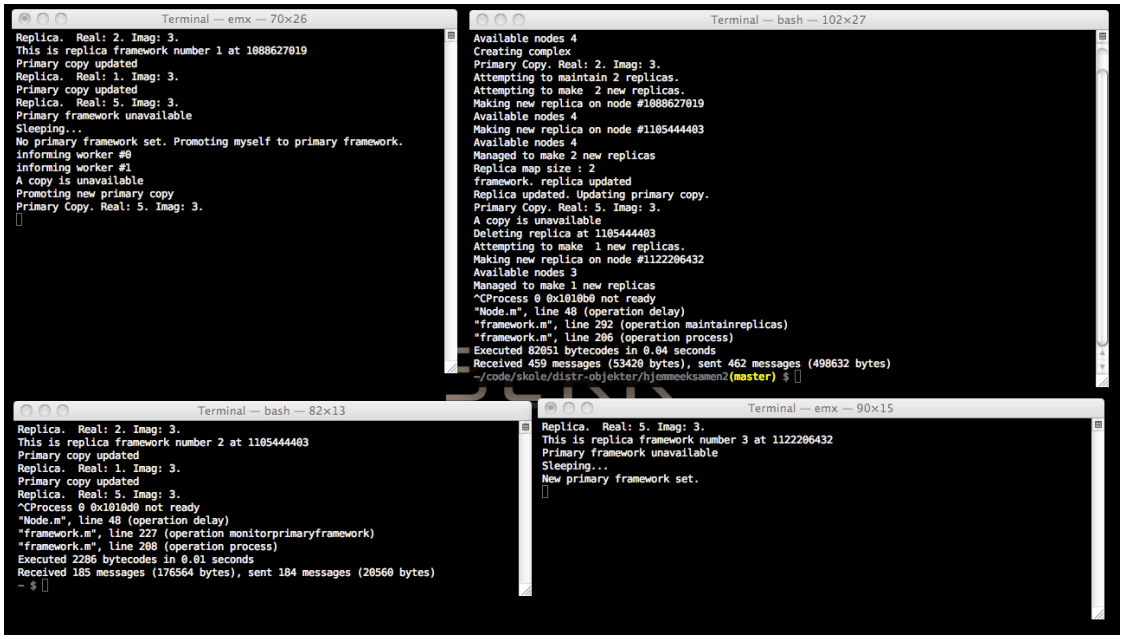
Oppdaterer en verdi på primary copy.

Venter fem sekunder

Oppdaterer en verdi på en av replikaene.

Så forsvinner en av nodene med replika på.

Så forsvinner noden med primary copy og primary framework.



Skjermbilde er vedlagt i zip-fil.