

Fil-synkronisering

Daniel Brixen

11. november 2002

Resumé

Et foredrag om fil-synkronisering baseret på artiklerne:

- What is a File Synchronizer?
- An Algebraic Approach to File Synchronization

Hvad er fil-synkronisering

- Data-replikering, hvor der udføres lokale opdateringer skaber behov for *synkronisering*
- Granularitet: fil-niveau
- Synligt for brugeren, at der er flere kopier af data
- Overordnet fremgangsmetode:
 - Opdate ændringer
 - Propagere ændringer (hvis der ikke er konflikter)

Oversigt over metode

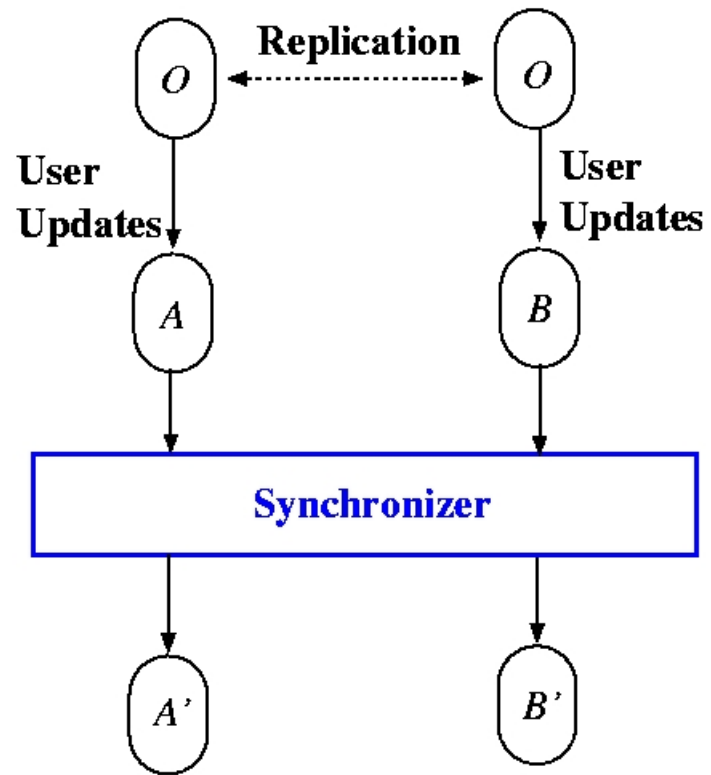
- Update detection

Dvs. identificere hvilke ændringer der er foretaget på en given replika siden sidste synkronisering.

- Reconciliation

De ændringer der er udført på de enkelte replikaer forsøges udført på samtlige replikaer, så alle replikaer er ens.

Oversigt over metode(2)



Definitioner

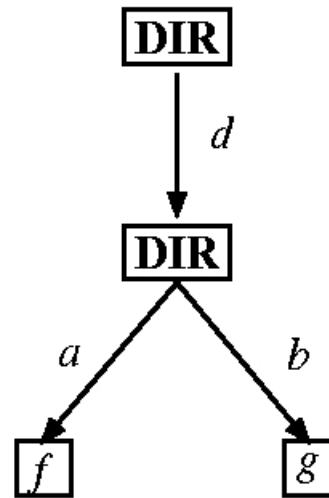
- Mængde af filnavne: N
- Mængde af stier: P - endelige sekvens af filnavne adskilt af “.”
- Mængde af filer F - fx strenge over et alfabet

- Filsystem:

$$FS = \{S \in P \rightarrow (F \cup \{DIR, \perp\}) \mid \forall p, x \in P : S(p.x) \neq \perp \Rightarrow S(p) = DIR\}$$

- For to stier p og q er q et *prefix* af p : $q \leq p : \exists r : p = q.r$.

Definitioner (eksempel)



$F = \{\epsilon \mapsto F, d \mapsto D, d.a \mapsto f, d.b \mapsto g, p \mapsto \perp \text{ for alle andre } p\}$

$D = \{\epsilon \mapsto D, d \mapsto D, a \mapsto f, b \mapsto g, p \mapsto \perp \text{ for alle andre } p\}$

Definitioner

- For filsystemer A og B , og en sti p :
 - $children_A(p) = \{q \mid q = p.x \wedge A(q) \neq \perp\}$
 - $children_{A,B}(p) = children_A(p) \cup children_B(p)$
- For filsystemet A og B , og en sti p :
 - $isDir_A(p)$ sand, netop hvis p er et bibliotek, dvs. ikke en fil, og ikke \perp .
 - $isDir_{A,B}(p) = isDir_A(p) \wedge isDir_B(p)$

Forsimplende antagelser

- Præcis 2 replikaer (A og B)
- Filsystemer statiske under synkroniseringen
- Efter forrige synkronisering var de to replikaer ens (tilstand O)
- Ignorerer filrettigheder og links

Update detection

- Krav til *dirty*-prædikatet:

Antag O og S filsystemer. Prædikatet $dirty_S$ skal for hver sti p opfylde, at hvis $\neg dirty_S(p)$ så er $O(p) = S(p)$

- Konsekvenser:

Hvis en sti p ikke er modificeret (*dirty*) i hverken A eller B , så har de to replikaer samme værdi i p .

Hvis $p \leq q$, og $dirty_S(q)$ så $dirty_S(p)$.

Update detection (2)

Metoder til at definere $dirty_S(p)$:

- Trivial: $dirty_S(p) = true$ for alle p
- Eksakt: $dirty_S(p) = true$ for netop de p , hvor der er udført en ændring
- Modifikationstid
- Modifikationstid og inode-nummer
- “On-line”

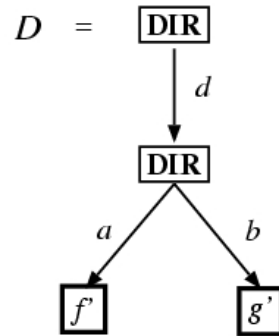
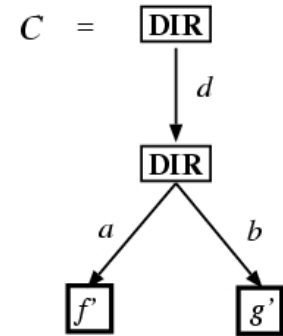
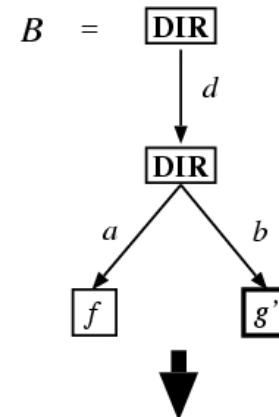
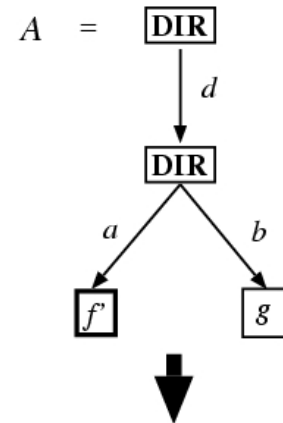
Reconciliation

Opgave:

- Propagere ændringer, hvor der ikke er konflikter
- Hvis der er en konflikt, så undlad at gøre noget

Der er en konflikt ved en sti p , hvis p er *dirty* i A og B , og det nye indhold ikke er ens.

Reconciliation (eksempel)



Specifikation af reconciliation

Der udføres reconciliation på de to replikaer A og B der antages at have fælles udgangstilstand O , hvilket producerer to nye tilstande C og D . Hvis følgende relationer er opfyldte for alle (relevante) stier p , er C og D en *synkronisering* af A og B . En sti p er *relevant* for (A, B) , hvis $\forall q < p : isDir_{A,B}(q)$.

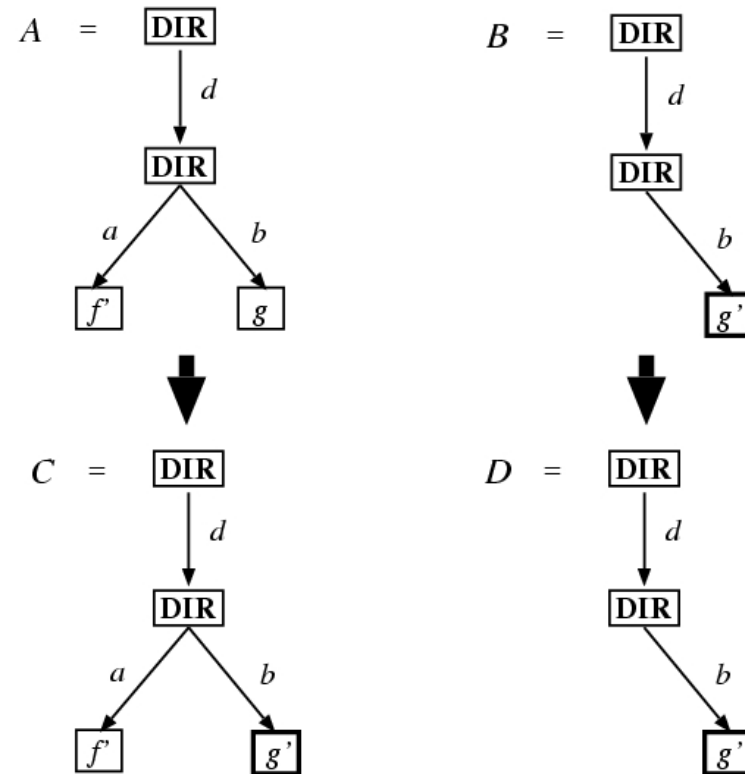
$$\neg dirty_A(p) \Rightarrow C(p) = D(p) = B(p)$$

$$\neg dirty_B(p) \Rightarrow C(p) = D(p) = A(p)$$

$$isDir_{A,B}(p) \Rightarrow isdir_{C,D}(p)$$

$$dirty_A(p) \wedge dirty_B(p) \wedge \neg isdir_{A,B}(p) \Rightarrow C(p) = A(p) \wedge D(p) = B(p)$$

Reconciliation (eksempel)



Algoritme

$recon(A, B, p) =$
 1) if $\neg dirty_A(p) \wedge \neg dirty_B(p)$
 then (A, B)
 2) else if $isdir_{A,B}(p)$
 then let $\{p_1, p_2, \dots, p_n\} = children_{A,B}(p)$
 (in lexicographic order)
 in let $(A_0, B_0) = (A, B)$
 let $(A_{i+1}, B_{i+1}) = recon(A_i, B_i, p_{i+1})$
 for $0 \leq i < n$
 in (A_n, B_n)
 3) else if $\neg dirty_A(p)$
 then $(A \stackrel{p}{\leftarrow} B, B)$
 4) else if $\neg dirty_B(p)$
 then $(A, B \stackrel{p}{\leftarrow} A)$
 5) else
 (A, B) .

Egenskaber ved specifikationen og algoritmen

- Algoritmen terminerer for alle A , B og p .
- Hvis $(C, D) = recon(A, B, \epsilon)$ så er (C, D) en synkronisering af A , og B (dvs. algoritmen lever op til specifikationen).
- Hvis (C, D) er en synkronisering af A og B , så er $recon(A, B, \epsilon) = (C', D')$ og $C' = C$ og $D = D'$ (dvs. algoritmen er ækvivalent med specifikationen).

En algebraisk tilgang

- *Operationer* i stedet for *tilstande*
- 3 faser i stedet for 2:
 - Update detection
 - Reconciliation
 - Conflict resolution

Model

Et filsystem F er en afbildning fra stier til filer og biblioteker.

- $F(p) = FILE(m, x)$ m : metadata, x : indhold
- $F(p) = DIR(m)$ m : metadata
- $F(p) = \perp$

F er *broken* ($F = \perp$) hvis der er udført en ulovlig operation på filsystemet.

Model (2)

Et filsystem F_1 siges at *approximere* F_2 :

$$F_1 \sqsubseteq F_2 \Leftrightarrow F_1 = \perp \vee (F_1 \neq \perp \wedge F_2 \neq \perp \wedge \forall p : F_1(p) = F_2(p))$$

Operationer på et filsystem

$$create(\pi, X)F = \begin{cases} F\{\pi \mapsto X\} & \text{hvis } F \neq \perp \wedge F(\pi) = \perp \wedge F(parent(\pi)) = DIR(\cdot) \\ \perp & \text{ellers} \end{cases}$$

$$edit(\pi, DIR(m))F = \begin{cases} F\{\pi \mapsto DIR(m)\} & \text{hvis } F \neq \perp \wedge F(\pi) \neq \perp \\ \perp & \text{ellers} \end{cases}$$

$$edit(\pi, FILE(m, x))F = \begin{cases} F\{\pi \mapsto FILE(m, x)\} & \text{hvis } F \neq \perp \wedge F(\pi) \neq \perp \wedge childless_F(\pi) \\ \perp & \text{ellers} \end{cases}$$

$$remove(\pi)F = \begin{cases} F\{\pi \mapsto \perp\} & \text{hvis } F \neq \perp \wedge F(\pi) \neq \perp \wedge childless_F(\pi) \\ \perp & \text{ellers} \end{cases}$$

$$breakF = \perp$$

Sekvenser af operationer

Vha. semikolon, kan der opbygges *sekvenser* af operationer (betegnes fx S_1). En sekvens er en funktion fra et filsystem til et andet. Fx:

$$(C_1; C_2)(F) = C_2(C_1(F))$$

Algebraiske love

Relationen \sqsubseteq defineres nu for sekvenser af operationer. Vi bruger \equiv som forkortelse:

$$S_1 \equiv S_2 \Leftrightarrow S_1 \sqsubseteq S_2 \wedge S_2 \sqsubseteq S_1$$

\sqsubseteq defineres ved at se på de forskellige muligheder for at kombinere to operationer, og udfra 4 inferensregler.

Sund og komplet

For at kunne ræsonere om filsystemerne ud fra sekvenser af operationer og relationen \sqsubseteq må vi kræve, at det opstillede bevissystem er *sundt* og *komplet*:

- *Sundt*:

$$S_1 \sqsubseteq S_2 \Rightarrow \forall F : S_1(F) \sqsubseteq S_2(F)$$

- *Komplet*:

$$(\forall F : S_1(F) \sqsubseteq S_2(F)) \Rightarrow S_1 \parallel S_2$$

hvor $S_1 \parallel S_2$ betyder, at der findes en sekvens S , så $S_1 \sqsubseteq S$ og $S_2 \sqsubseteq S$.

Sund og komplet (2)

Beviset for at systemet er komplet er konstruktivt: der konstrueres en *kanonisk sekvens* som både S_1 og S_2 approksimerer. Den har følgende struktur:

- $edit(\pi, DIR)$ kommandoer
- $create(\pi, X)$ kommandoer
- $remove(\pi)$ kommandoer
- $edit(\pi, FILE(m, x))$ kommandoer

Update detection

Update detection skal identificere en sekvens af operationer der kan være udført på filsystemet O , så det kommer i tilstand A (eller B).

Der er mange muligheder, så stræb efter en med færrest antal operationer.

Lad S'_1 være den sekvens af operationer der identificeres som værende udført på den første replika, og tilsvarende S'_2 . Disse to sekvenser omordnes til *kanoniske sekvenser* S_1 og S_2 , der ifølge komplet-heden kan bruges i stedet for S'_1 og S'_2 .

Reconciliation

Målet med reconciliation er at finde sekvenser S_1^* og S_2^* , så

$$F_{\text{efter}} = S_1^*(S_1(O)) = S_2^*(S_2(O))$$

En kommando $C \in S_1$ skal propageres til den anden replika (dvs. inkluderes i sekvensen S_2^*) netop hvis:

- $C \notin S_2$, dvs. hvis C ikke allerede er udført på den anden replika
- Ingen operationer fra S_2 er i konflikt med C .
- Ingen operationer fra S_2 er i konflikt med operationer der skal komme før C .

Reconciliation (konflikt)

To operationer $C_i(\pi) \in S_1$ og $C_j(\gamma) \in S_2$ er i *konflikt* hvis

- $C_i(\pi) \notin S_1$ og $C_j(\pi) \notin S_2$ og
- $C_i(\pi); C_j(\gamma) \not\equiv C_j(\gamma); C_i(\pi)$, dvs. operationerne kommuterer ikke, eller
- $C_i(\pi); C_j(\gamma) \equiv \text{break}$ eller
- $C_j(\gamma); C_i(\pi) \equiv \text{break}$

Reconciliation (metode)

Operationerne fra sekvenserne S_1 og S_2 behandles nu efter tur. Først undersøges det, om operationerne fra S_1 skal tilføjes til S_2^* , og derefter om operationerne fra S_2 skal tilføjes til S_1^* .

Konflikt-løsning

Efter reconciliation er der muligvis operationer fra S_1 og S_2 der ikke er tilføjet S_2^* hhv. S_1^* . Konflikt-løsning går ud på at bestemme, hvad der skal ske med disse operationer.

- Se bort fra operationer der er i konflikt

Dette svarer til strategien der bruges i den tilstandsorienterede tilgang

- Propager information om konflikter til alle replikaer

Muliggør reparation af en replika uden kontakt med de andre replikaer

- Foretag ændringer i de operationer der er i konflikt, så der ikke længere er konflikter. På denne måde kan replikaerne være ens efter synkronisering.

Spørgsmål

- I “What is a File Synchronizer” antages det, at de to replikaer var i samme tilstand efter forrige synkronisering. Hvilke ændringer skal der til for at undlade denne antagelse? (altså at der skal kunne foretages synkronisering på to replikaer A' og B' , hvis starttilstande var hhv. A og B og $A \neq B$)
- Foreslå en metode hvorved den tilstands-baserede metode kan bruges til at foretage synkronisering af mere end to replikaer.
- Hvordan kan man finde en sekvens af operationer der fører filsystemet F til filsystem F' ?