Sistem distribuit de sincronizare al fișierelor

Îndrumător: Ş.l., dr., ing., Cristian-Mihai Amarandei

Student: Andrei Chelariu, 1408A

1. Descrierea funcționalității programului

1.1 Scop

Programul are ca scop sincronizarea unei ierarhii de fișiere în cadrul unui sistem distribuit. Atunci când un utilizator face o modificare asupra sistemului de fișiere, modificarea lui este transmisa şi celorlalți utilizatori din cadrul sistemului.

1.2 Cazuri de utilizare

Utilizatorul poate sa activeze sistemul sau sa îl dezactiveze.

Pentru activarea sistemului, utilizatorul trebuie sa precizeze directorul în care va fi salvată ierarhia sistemului şi directorul în care vor fi salvate tentativele(fişierele modificate de utilizatorul curent, dar care nu au fost validate de sistem). Ambele directoare trebuie să fie goale. Conținutul lor va fi șters la inițializarea sistemului.

La dezactivarea sistemului, utilizatorul notifica restul sistemului distribuit ca nu mai este interesat de structura sistemului de fișiere. Dacă utilizatorul va dori sa activeze sistemul la un alt moment în timp, el va trebui sa descarce din nou toată structura. Se face o deosebire intre întreruperea involuntara a sistemului şi dezactivarea lui de către utilizator.

1.3 Principiu de funcționare

Funcționalitatea sistemului poate fi descrisa în următoarele faze:

1. Utilizatorul face modificări asupra structurii de fișiere. Modificările vor fi detectate după ce s-au petrecut. Pentru fiecare modificare se va genera și o operație de refacere, *undo*.
2. Modificările utilizatorului sunt validate de către un proces central care se ocupa de controlul consistentei. Modificările utilizatorului vor fi marcate ca tentative şi vor fi salvate în fişierul de tentative. Dacă modificările nu sunt validate, starea sistemului va fi refăcuta, iar utilizatorul va fi notificat.
3. Dacă modificările sunt validate, acestea sunt transmise către ceilalți utilizatori ai sistemului. Dacă transmiterea modificărilor eşuează, starea sistemului va fi refăcuta, iar utilizatorul va fi notificat.
4. Dacă modificările sunt transmise cu succes, copiile tentativelor sunt șterse din directorul de tentative, iar starea sistemului este din nou stabilă.

1.4 Constrângeri

Principalele constrângeri ale sistemului sunt:

1. Atomicitate: o modificare a sistemului de fișiere trebuie sa ajungă la toți utilizatorii sau la nici unul.
2. Consistenta: sistemul trebuie sa rămână valid după efectuarea unei operații

3. Izolare: doua operații în conflict nu pot fi efectuate în același timp de doi utilizatori diferiți.
4. Durabilitate: modificările făcute trebuie să facă parte din structura sistemului de fișiere.
5. Aplicația trebuie sa ruleze pe sistemul de operare Linux.
6. O componenta care s-a defectat poate fi repornita, fără a reporni toată aplicația.
7. Sistemul trebuie să rămână consistent în cazul în care un utilizator iese din sistem(intenționat sau în urma unei defecțiuni).
8. Toate operațiile făcute asupra sistemului de fișiere trebuie sa aibă o operație de refacere.

2. Arhitectura aplicației

2.1 Descrierea componentelor

2.1.1 Passive front end

Rol: Componenta passive front end este responsabila cu detectarea modificărilor la nivelul sistemului de fișiere. În acest scop, ea se va înregistra la nivelul sistemului de operare pentru detectarea modificărilor făcute de utilizator.

Comunicarea cu alte componente: Modificările făcute de utilizator sunt transmise printr-un mecanism inter proces, cum ar fi *UNIX domain sockets* sau *named pipes* către o componenta care va consuma aceste evenimente.

Tehnologii utilizate:

- C/C++ sau lua sau Python
- UNIX domain sockets sau named pipes
- sql lite

2.1.2 Replication manager

Rol: Componenta replication manager este responsabila de replicarea modificărilor generate de utilizator.

Descrierea funcționalității: Componenta replication manager consuma evenimentele generate de passive front end. Atunci când un eveniment este recepționat, replication manager-ul comunica cu central consistency manager, pentru a stabili dacă modificările sunt valide. Dacă modificările nu sunt validate, atunci replication manager-ul copie fișierele modificate de utilizator în fișierul de tentative și executa operațiile de refacere. În caz de succes, replication manager-ul folosește un sistem de broadcast pentru a transmite modificările către ceilalți utilizatori din sistem.

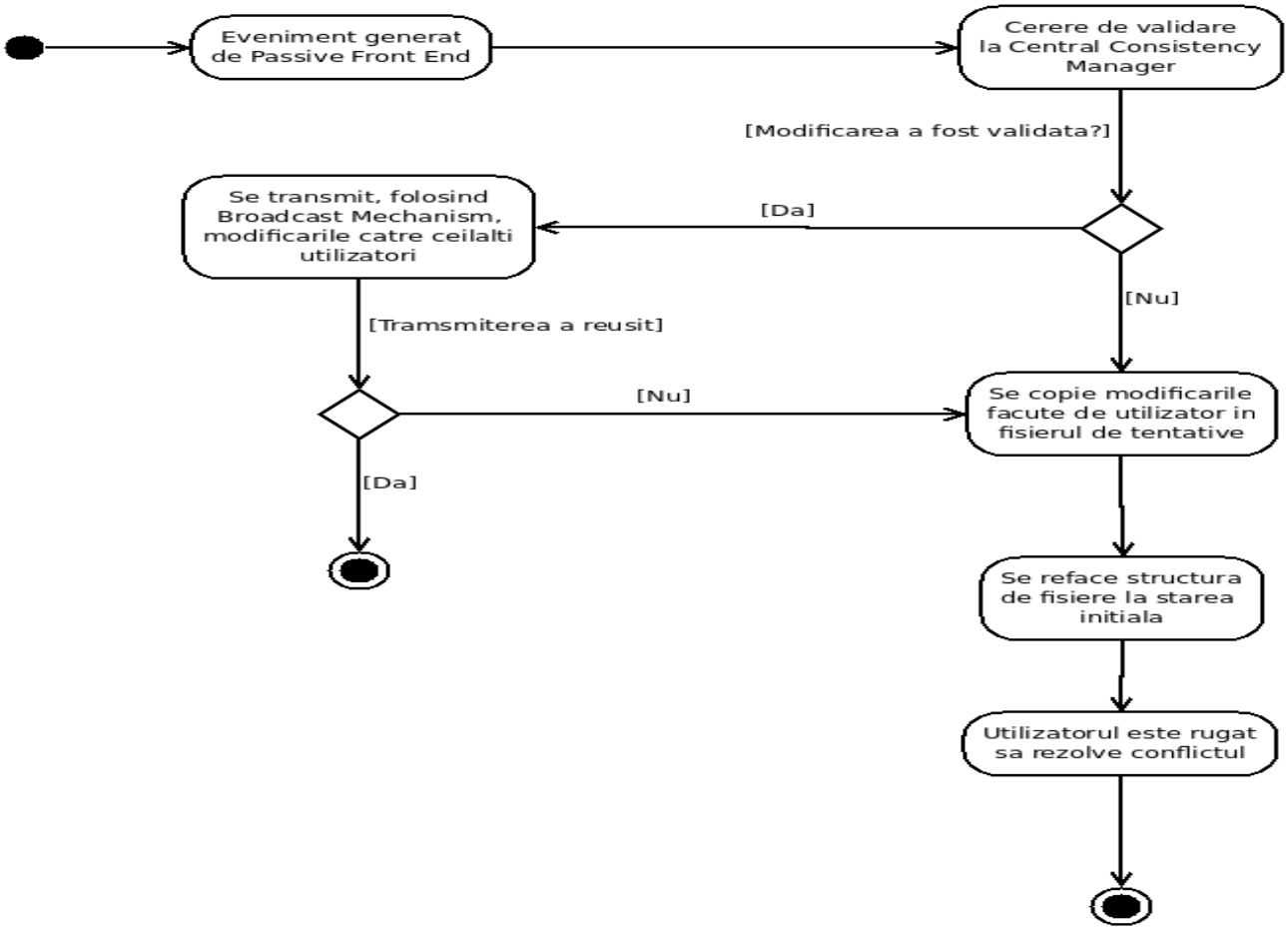
Comunicarea cu alte componente:

- Passive front end, folosind mecanisme IPC
- Central Consistency Manager, folosind mesaje în rețea(protocol propriu sau HTTP)
- Broadcast System, prin folosirea unui DLL sau prin interfața de loopback.

Tehnologii utilizate:

- Python
- biblioteca de networking Twisted
- sql lite

Diagrama de activități:



2.1.3 Central consistency manager

Rol: Asigurarea consistentei în cadrul sistemului distribuit.

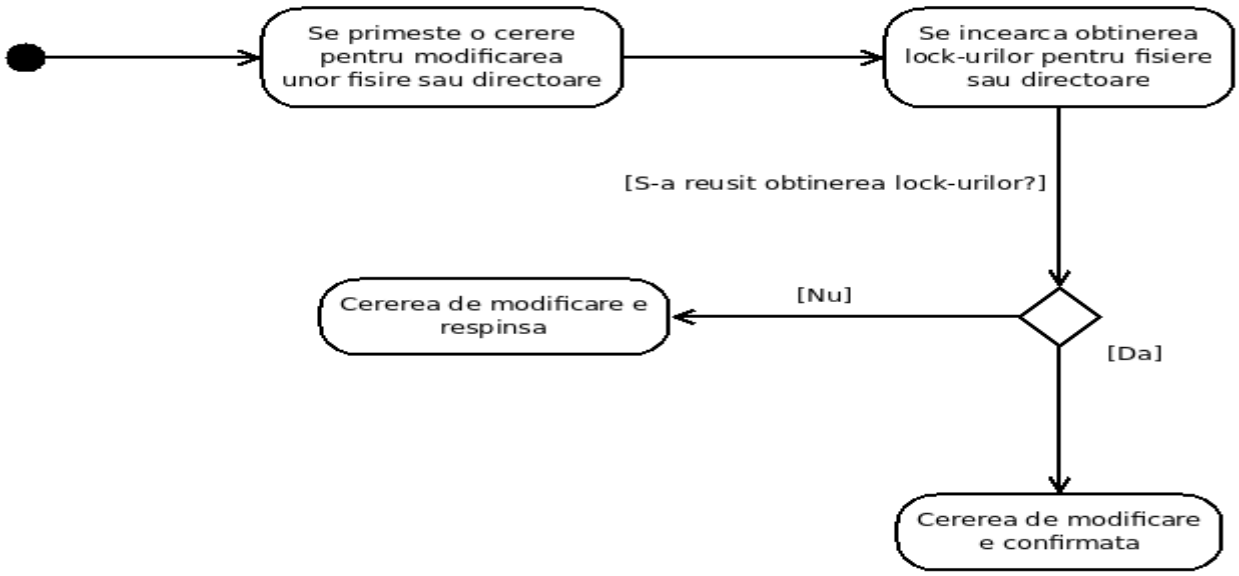
Descrierea funcționalității: Pentru fiecare fișier și fiecare director componenta va menține câte un lock. În momentul în care replication managerul accesează operația de validare, central consistency managerul va marca lock-urile corespunzătoare ca luate. Dacă lock-urile sunt deja luate, atunci operația este invalida. În caz de succes, se returnează un mesaj de confirmare. Mecanismul de control al concurentei este unul optimist.

Comunicarea cu alte componente: replication manager, folosind mesaje în rețea. Protocolul de comunicare va fi unul propriu sau protocolul HTTP.

Tehnologii utilizate:

- C++ și BOOST sau Python și Twisted

Diagrama de activități:



2.1.4 Broadcast mechanism

Rol: Asigura comunicarea în grup a clienților.

Constrângeri:

- mesajele trebuie sa ajungă la toți clienții sau la nici unul
- mesajele trebuie sa ajungă exact o data la fiecare client
- mesajele trebuie să fie ordonate

Tehnologii utilizate:

- ZEROMQ

Observații: Deoarece un sistem de broadcast este foarte complex și nu am timp suficient, voi folosi sistemul ZEROMQ.

2.2 Descrierea interacțiunii dintre componente

Cele patru componente din cadrul sistemului comunica fie prin elemente IPC, fie prin rețea.

Am optat pentru separarea fiecărei componente în procesul ei propriu din următoarele motive:

1. Cuplarea slaba: cele patru componente trebuie să fie independente intre ele. O componenta va putea fi rescrisa pe viitor, fără a fi necesara rescrierea întregii aplicații. De exemplu, intenționez ca pe viitor sa realizez propriul meu sistem de broadcast folosind arhitectura gossip, dar restul aplicației sa rămână la fel.
2. Independenta în caz de erori: în cazul în care o componenta se va defecta, restul aplicației va trebui sa funcționeze. Acest lucru nu va putea fi realizat în totalitate.
3. Independenta la nivel de limbaj: cele 4 componente pot fi realizate folosind tehnologii diferite, fiecare având propriile avantaje. De exemplu, Central Consistency Manager poate fi realizat folosind C++ pentru o mai buna performață, iar Replication Managerul poate fi

realizat în Python pentru mai multă flexibilitate.

4. Paralelism: deoarece cele 4 componente sunt procese de sine stătătoare, ele pot fi executate în paralel de către nucleele procesorului. Astfel, paralelizarea se realizează în mod implicit, fără a fi nevoie de lucru direct cu thread-uri.

Diagrama componentelor este prezentata în figura de mai jos:

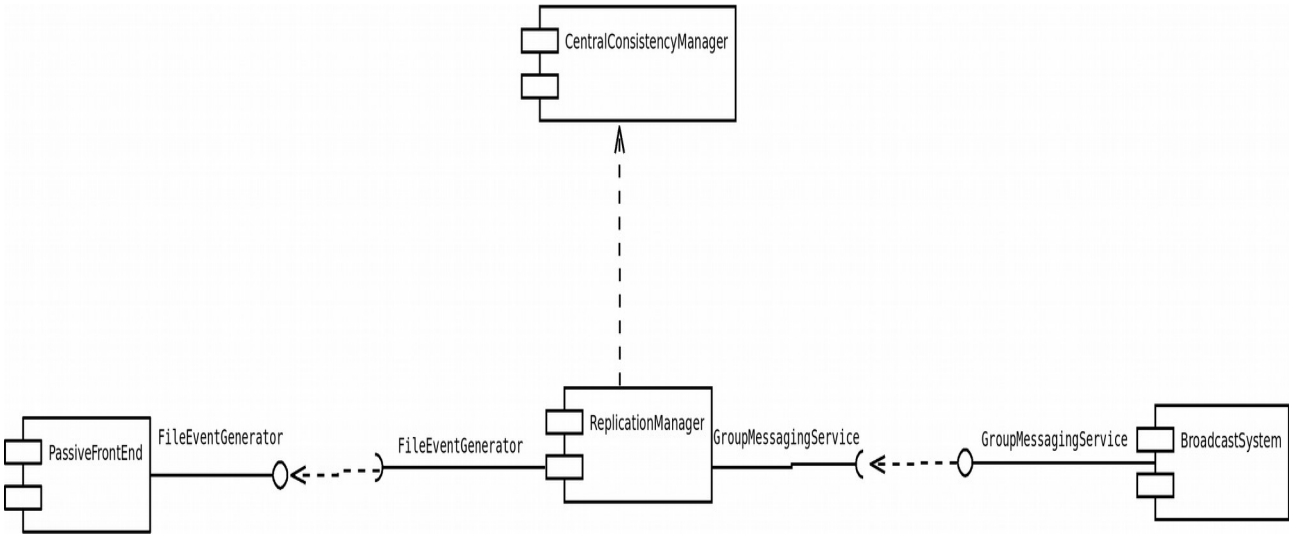


Diagrama de secvența este prezentata în figura de mai jos: