Proiect Sisteme Distribuite

Salnicean Razvan

19 ianuarie 2026

Cuprins

1 Introducere

Am dezvoltat o platformă de trading distribuită care permite utilizatorilor să tranzacționeze acțiuni și criptomonede într-un mediu peer-to-peer. Sistemul pe care l-am creat utilizează o arhitectură bazată pe evenimente și comunicare multicast pentru a sincroniza datele între mai multe instanțe ale aplicației care rulează pe diferite mașini din rețea.

Ideea de bază a fost să creez un sistem unde mai mulți utilizatori pot participa simultan la piață, fie tranzacționând între ei prin ordine limită, fie cumpărând/vânzând direct de la o entitate centrală pe care am numit-o "Bank". Prețurile sunt actualizate în timp real prin conectare la API-uri externe pentru acțiuni (Stooq) și criptomonede (Binance).

1.1 Motivație

Am vrut să construiesc ceva ce combină mai multe concepte interesante: programare distribuită, sisteme bazate pe evenimente, integrare cu API-uri externe și un domeniu aplicativ care să fie relevant. Trading-ul oferă toate acestea și mai adaugă și complexitatea de a gestiona ordine, matching-ul acestora și sincronizarea stării între mai mulți participanți.

1.2 Obiective

Obiectivele pe care mi le-am propus pentru acest proiect au fost:

- Să creez un sistem distribuit funcțional folosind Java și comunicare multicast
- Să integrez date de piață în timp real de la surse externe
- Să implementez un motor de matching pentru ordine limită între utilizatori
- Să permit și tranzacții directe la prețul pieței
- Să oferă o interfață interactivă prin linia de comandă

2 Arhitectura Sistemului

Arhitectura se bazează pe un model publicare-abonare (publish-subscribe) cu sincronizare prin multicast UDP. Această abordare permite scalabilitate și decuplare între componente.

2.1 Componente Principale

Sistemul este format din mai multe componente care colaborează pentru a oferi funcționalitatea completă.

2.1.1 Event Bus Distribuit

Clasa `DistributedEventBus` reprezintă nucleul sistemului și gestionează comunicarea între noduri și distribuirea evenimentelor. Fiecare nod din rețea are propriul event bus care:

- Publică evenimente local către listenerii înregistrați

- Trimite evenimente prin rețea către celelalte noduri
- Recepționează evenimente de la alte noduri și le procesează local

Fiecare nod are un identificator unic (UUID) pentru a putea filtra propriile mesaje când acestea sunt primite înapoi prin multicast.

2.1.2 Serviciul de Rețea

Clasa `NetworkService` gestionează comunicarea de nivel jos, folosind un `MulticastSocket` pentru trimiterea și recepționarea pachetelor UDP. Multicast-ul a fost ales pentru că permite comunicarea one-to-many eficientă - un mesaj trimis ajunge la toate nodurile interesate fără a fi necesară cunoașterea adreselor individuale.

Evenimentele sunt serializate folosind mecanismul Java de serializare pentru a fi transformate în bytes care pot fi trimiși prin rețea. La recepție, procesul invers deserializează bytes-urile înapoi în obiecte.

2.2 Tipuri de Evenimente

Sistemul definește mai multe tipuri de evenimente pentru diferite operațiuni:

- **PriceUpdateEvent** - actualizări de preț pentru simboluri
- **OrderEvent** - ordine limită între utilizatori (P2P)
- **DirectOrderEvent** - ordine directe cu Bank-ul
- **TradeExecutedEvent** - confirmări de tranzacții executate
- **SymbolRequestEvent** - cereri de adăugare simboluri noi
- **AvailableSymbolsRequestEvent/ResponseEvent** - listarea simbolurilor disponibile

Toate aceste clase moștenesc din `TradingEvent` și implementează `Serializable` pentru a putea fi trimise prin rețea.

3 Serverul de Piață

Serverul este implementat în clasa `MarketServer` și are mai multe responsabilități importante în cadrul sistemului.

3.1 Gestionarea Prețurilor

Serverul menține o hartă cu prețurile curente pentru toate simbolurile active. La pornire, sunt configurate câteva simboluri populare să înceapă automat (AAPL.US pentru acțiuni și BTCUSDT pentru crypto). Utilizatorii pot cere adăugarea de simboluri noi prin `SymbolRequestEvent`.

Pentru fiecare simbol activ, este pornit un thread separat care rulează `RealTimePriceGenerator`. Acesta face polling periodic la API-ul corespunzător și publică evenimente `PriceUpdateEvent` când găsește un preț nou.

3.2 Integrarea cu API-uri Externe

Există două implementări ale interfeței `PriceFetcher` pentru diferite surse de date.

3.2.1 StooqStockFetcher

Pentru acțiuni, se folosește API-ul de la Stooq. Se face o cerere HTTP și se parsează CSV-ul returnat pentru a extrage prețul de închidere. URL-ul accesat are formatul:

`https://stooq.com/q/l/?s=SIMBOL&f=sd2t2ohlc&h&e=csv`

Simbolurile pentru acțiuni conțin de obicei un punct (ex: AAPL.US), ceea ce ajută la identificarea lor automată.

3.2.2 BinanceCryptoFetcher

Pentru criptomonede, se folosește API-ul public Binance. Endpoint-ul este mai simplu și returnează JSON:

`https://api.binance.com/api/v3/ticker/price?symbol=SIMBOL`

Se face un parsing simplu al JSON-ului pentru a extrage câmpul "price". Criptomonedele sunt identificate prin absența punctului în simbol (ex: BTCUSDT).

3.3 Directorul de Piață

Sistemul include un director care listează toate simbolurile disponibile. La inițializare, serverul:

1. Adaugă o listă hardcodată de acțiuni populare (AAPL, MSFT, GOOGL, etc.)
2. Interoghează API-ul Binance pentru a obține toate perechile USDT disponibile
3. Limitează la 100 de simboluri crypto pentru a nu supraîncărca pachetele UDP

Când un client trimite `AvailableSymbolsRequestEvent`, serverul răspunde cu lista completă prin `AvailableSymbolsResponseEvent`.

4 Motorul de Matching

Una dintre componentele mai complexe este `OrderMatchingEngine`, care implementează logica de potrivire a ordinelor limită între utilizatori.

4.1 Structura de Date

Pentru fiecare simbol, se mențin două priority queues:

- **buyOrders** - sortată descrescător după preț (cei care oferă mai mult sunt primii)
- **sellOrders** - sortată crescător după preț (cei care cer mai puțin sunt primii)

Priority queues au fost alese pentru că permit găsirea rapidă a celei mai bune oferte/-cereri.

4.2 Algoritmul de Matching

Procesul de potrivire a unui ordin nou funcționează astfel:

1. Ordinul este adăugat în book-ul corespunzător (buy sau sell)
2. Se verifică dacă există potriviri - cel mai bun buy trebuie să aibă preț ¿= cel mai bun sell
3. Dacă condiția este îndeplinită, ambele ordine sunt scoase și se execută tranzacția
4. Prețul de execuție este media celor două prețuri
5. Cantitatea executată este minimul dintre cele două cantități
6. Se emite un `TradeExecutedEvent` cu detaliile

Există și o verificare pentru a preveni ca utilizatorii să tranzacționeze cu ei înșiși (verificare pe `userId`).

4.3 Ordine Directe

Pentru simplitate, sistemul include și ordine directe unde utilizatorii pot cumpăra/vinde instant la prețul curent de piață de la "Bank". Acestea sunt procesate imediat fără matching, folosind ultimul preț cunoscut pentru simbol.

5 Clientul de Trading

Clasa `TraderClient` oferă interfața pentru utilizatori. Fiecare client are:

- Un `userId` unic
- Un portofoliu (map de simboluri la cantități)
- Un sold în numerar (inițial 10000)
- O hartă locală cu prețurile de piață primite

5.1 Gestionarea Evenimentelor

Clientul se abonează la mai multe tipuri de evenimente pentru a reacționa la schimbările din sistem.

5.1.1 PriceUpdateEvent

Când se primesc actualizări de preț, acestea sunt stocate local în map-ul `marketPrices`. Dacă modul "watch" este activ, prețul este afișat direct în consolă pentru feedback live.

5.1.2 TradeExecutedEvent

Aici se implementează logica de actualizare a portofoliului și soldului. Se verifică dacă utilizatorul curent este buyer sau seller și:

- Dacă este buyer: se scad banii din sold și se adaugă acțiuni în portofoliu
- Dacă este seller: se adaugă banii în sold și se scad acțiuni din portofoliu

Sunt afișate și mesaje clare în consolă pentru a înștiința când se execută tranzacții.

5.2 Interfața Interactivă

Interfața în linia de comandă acceptă următoarele comenzi:

- **available** - cere lista de simboluri de la server
- **add SIMBOL** - cere serverului să înceapă tracking pentru un simbol nou
- **ticker** - afișează snapshot cu toate prețurile curente
- **watch** - streamează prețurile live până se apasă Enter
- **buy SIMBOL PRET CANTITATE** - plasează ordin limită P2P
- **sell SIMBOL PRET CANTITATE** - plasează ordin de vânzare P2P
- **buy-direct SIMBOL CANTITATE** - cumpără instant de la Bank
- **sell-direct SIMBOL CANTITATE** - vinde instant la Bank
- **port** - afișează portofoliul și soldul
- **help** - lista de comenzi

Există și validări pentru a preveni vânzarea de acțiuni care nu sunt deținute. Dacă cineva încearcă să vândă mai mult decât are, sistemul afișează o eroare în română.

6 Comunicarea în Rețea

Aspectul distribuit al sistemului se bazează pe comunicarea multicast UDP.

6.1 Configurarea Multicast

Se folosește adresa multicast `230.0.0.0` și portul `8888`. În metoda `findBestInterface` este implementată logica pentru a găsi interfața de rețea optimă:

1. Se iterează prin toate interfețele disponibile
2. Se sar peste cele de loopback sau dezactivate
3. Se caută prima interfață cu adresă IPv4
4. Dacă nu se găsește niciuna, se folosește interfața locală default

Loopback mode este dezactivat pentru a primi și propriile mesaje (necesar pentru debugging), dar acestea sunt filtrate ulterior după UUID.

6.2 Serializarea Evenimentelor

Pentru a trimite obiecte prin rețea, se folosește:

```
ByteArrayOutputStream + ObjectOutputStream
```

La recepție, procesul invers:

```
ByteArrayInputStream + ObjectInputStream
```

Toate excepțiile sunt prinse și logate fără a opri thread-urile de ascultare.

6.3 Filtrarea Mesajelor

Pentru a evita procesarea propriilor mesaje, fiecare eveniment are setat `originNodeId`. La recepție, se verifică dacă UUID-ul din eveniment coincide cu cel local și, dacă da, mesajul este ignorat.

7 Detalii de Implementare

7.1 Concurență și Thread Safety

Sunt folosite mai multe mecanisme pentru a asigura thread safety în sistem.

7.1.1 ConcurrentHashMap

Majoritatea map-urilor sunt `ConcurrentHashMap` pentru acces concurrent sigur:

- `currentPrices` - prețurile de piață
- `portfolio` - portofoliul clientului
- `marketPrices` - prețurile locale ale clientului
- `listeners` - listenerii înregistrați în event bus

7.1.2 CopyOnWriteArrayList

Pentru colecțiile de listeneri și directorul de piață, se folosește `CopyOnWriteArrayList` care este optimizat pentru operații de citire frecvente și scrieri rare.

7.1.3 Sincronizare Explicită

În motorul de matching, se folosește `synchronized` pe metodele critice pentru a preveni race conditions când se potrivesc ordine simultan.

7.1.4 AtomicBoolean

Pentru flag-ul de watching în client, se folosește `AtomicBoolean` pentru actualizări thread-safe.

7.2 Gestionarea Thread-urilor

Sunt create mai multe tipuri de thread-uri în sistem:

1. **NetworkListener** - thread dedicat pentru primirea pachetelor UDP
2. **CachedThreadPool** - pentru executarea asincronă a listener-ilor
3. **RealTimePriceGenerator threads** - câte unul pentru fiecare simbol tracked
4. **Directory loader thread** - pentru încărcarea asincronă a directorului

7.3 Tratarea Erorilor

Tratarea erorilor este implementată la mai multe niveluri pentru robustețe.

7.3.1 Erori de Rețea

Când se trimit/primesc pachete, `IOException` este prins și mesajul este logat fără a opri aplicația. Pierderea ocazională de pachete este considerată acceptabilă în UDP.

7.3.2 Erori de Fetching

Când nu se poate obține un preț de la API, eroarea este logată dar thread-ul continuă cu următoarea încercare după interval. Astfel sistemul devine rezistent la probleme temporare de conexiune.

7.3.3 Validări de Business Logic

Există validări pentru:

- Vânzarea de cantități mai mari decât cele deținute
- Comenzi cu parametri lipsă sau incorecți
- Simboluri fără preț disponibil

8 Moduri de Rulare

Sistemul poate rula în două moduri principale, în funcție de parametrii primiți.

8.1 Mod Server

Pornit cu:

```
java StockTradingPlatform server
```

În acest mod, aplicația:

- Inițializează `MarketServer`
- Începe tracking-ul simbolurilor default

- Încarcă directorul de piață
- Procesează cereri de la clienți
- Rulează până la Ctrl+C

8.2 Mod Client

Pornit cu:

```
java StockTradingPlatform client USERNAME
```

sau interactiv (va cere username-ul dacă nu e furnizat).

Clientul oferă interfața de comandă pentru utilizator și se conectează automat la rețeaua multicast pentru a comunica cu serverul și ceilalți clienți.

9 Flux de Lucru Tipic

Un scenariu real de utilizare al sistemului arată astfel.

9.1 Inițializare

1. Se pornește serverul pe o mașină
2. Serverul începe polling la API-uri pentru AAPL.US și BTCUSDT
3. Serverul încarcă directorul cu toate simbolurile disponibile

9.2 Conectarea Clienților

1. Se pornește un client cu username "Alice"
2. Alice primește automat update-uri de preț pentru simbolurile active
3. Alice poate cere lista completă cu comanda "available"

9.3 Tranzacționare P2P

1. Alice: `buy AAPL.US 150.0 10` (dorește să cumpere 10 acțiuni la max 150)
2. Bob (alt client): `sell AAPL.US 149.0 5` (vinde 5 acțiuni la min 149)
3. Motorul de matching observă că 150 ¿= 149
4. Se execută tranzacție: 5 acțiuni la (150+149)/2 = 149.5
5. Ambii primesc `TradeExecutedEvent`
6. Portofoliile și soldurile se actualizează automat

9.4 Tranzacționare Directă

1. Alice: `buy-direct BTCUSDT 20`
2. Serverul verifică prețul curent (să zicem 45000)
3. Se execută instant: Alice cumpără 20 BTC la 45000 fiecare
4. Alice primește actualizarea portofoliului

9.5 Adăugare Simboluri

1. Alice: `add NVDA.US`
2. Serverul primește cererea
3. Se pornește thread nou pentru NVDA.US cu StooqStockFetcher
4. Toți clienții încep să primească update-uri pentru NVDA.US

10 Limitări și Îmbunătățiri Posibile

Sistemul actual prezintă câteva limitări care pot fi abordate în versiuni viitoare.

10.1 Limitări Curente

- Nu există persistență a datelor între restartări
- Lipsește autentificarea sau securitatea
- UDP poate pierde pachete în rețele congeste
- Order matching-ul este basic (nu suportă ordine parțiale)
- Nu există time-in-force pentru ordine
- Rate limiting rudimentar pentru API-uri externe

10.2 Îmbunătățiri Posibile

Dezvoltarea viitoare ar putea include:

1. **Persistență** - salvarea portofoliilor în bază de date
2. **Ordine avansate** - stop-loss, take-profit, trailing stop
3. **Istoric** - tracking complet al tuturor tranzacțiilor
4. **Grafice** - vizualizarea prețurilor în timp real
5. **Algoritmi de trading** - boti care tranzacționează automat
6. **WebSocket API** - interfață web în loc de CLI
7. **Autentificare** - sistem de login și permisiuni
8. **Multi-currency** - suport pentru mai multe valute fiat

11 Concluzii

Platforma de trading distribuită demonstrează implementarea unei arhitecturi event-driven folosind Java și comunicare multicast. Sistemul combină mai multe tehnologii și concepte pentru a oferi o soluție funcțională de tranzacționare peer-to-peer.

Aspectele cheie ale implementării includ:

- Sincronizarea stării în sisteme distribuite prin evenimente
- Thread safety când se lucrează cu date partajate
- Integrarea API-urilor externe în timp real
- Tratarea elegantă a erorilor în sisteme asincrone

Platforma oferă o bază solidă care poate fi extinsă cu funcționalități suplimentare. Codul este modular și poate fi adaptat pentru alte cazuri de utilizare care necesită comunicare distribuită bazată pe evenimente. Arhitectura aleasă permite scalabilitatea sistemului și adăugarea de noi features fără modificări majore ale structurii existente.