REPUBLIKA E SHQIPERISË

UNIVERSITETI POLITEKNIK I TIRANËS
FAKULTETI I TEKNOLOGJISË SË INFORMACIONIT

Inxhinieri informatike

Detyrë Kursi

Lënda: Programim në Sisteme të Shpërndara

Tema: Apache Zookeeper

Grupi: III-B

Punoi: Edison Zyberaj, Kejdi Shahu, Piro Gjikdhima

Pranoi: Megi Tartari

Viti Akademik 2024 - 2025

Detyra

Tema: Apache Zookeeper

Projektoni dhe ndertoni sistemet e meposhtme duke perdorur Apache Zookeeper. Dorezoni implementimet tuaja ne formatin ZIP, si dhe dokumentimin e implementimit te sistemeve, duke pershkruar qarte hapat per te arritur rezultatet e deshiruara.

Cluster Node Monitoring

Krijoni nje cluster me te pakten 3 Zookeeper nodes. Implementoni nje sistem monitorimi per te ndjekur statusin e çdo node ne cluster. Shkruani nje raport te shkurter qe shpjegon dizajnin e sistemit te monitorimit, duke perfshire metrikat qe zgjodhet per te monitoruar dhe si menaxhohen node failures.

Distributed Locking

Ndertoni nje sistem te shperndare qe perdor Zookeeper per te implementuar distributed locking. 2. Perdorni Apache Curator ose API-te native te Zookeeper per te krijuar dhe menaxhuar locking. 3. Shkruani nje raport qe pershkruan me detaje si implementimi juaj siguron mutual exclusion dhe si menaxhon edge cases.

Distributed Queues

Ndertoni nje sistem te shperndare qe ngre nje distributed queue mbi Zookeeper. Sistemi duhet te ofroje mundesine e futjes se elementeve ne queue nga disa Producers dhe leximin e tyre nga disa Consumers. 2. Testoni systemin tuaj me skenare te ndryshem si concurrent producers/consumers, network partition, dhe node failures. 3. Dokumentoni implementimin tuaj, duke shpjeguar optimizimet e bera per te permiresuar performancen dhe tolerancen ndaj gabimeve (fault tolerance).

Përdorimi i Docker për Zookeeper

Ky është një konfigurim Docker për ngritjen e një klasteri me 5 serverë Zookeeper. Po të analizojmë rresht për rresht, kuptojmë rolin e secilit pjesëtar dhe pse është e nevojshme kjo arkitekturë.

Ky konfigurim përmban 5 shërbime të ndara, nga zk1 deri zk5, secili prej të cilëve është një nodë Zookeeper.

Për çdo zk shërbim kemi:

- image: zookeeper:3.8 – përdoret imazhi zyrtar Docker i Zookeeper.
- container_name dhe hostname – përcaktojnë emrin e kontejnerit dhe hostit (që përputhen).
- ports – hapin portin 2181 të secilit node në portin lokal 2181, 2182, 2183, etj. për ta monitoruar ose përdorur në mënyrë të ndarë.
- environment:
    - ZOO_MY_ID – çdo nod ka një ID unike, nga 1 deri në 5.
    - ZOO_SERVERS – lista e të gjithë serverëve në klasër, në formatin:

Kjo është thelbësore për formimin e klasërit.

    - ZOO_4LW_COMMANDS_WHITELIST – lejon komanda si stat, conf, etj. për monitorim nga klienti.

networks:

zk-net:

external: true

Kjo tregon që të gjithë kontejnerët do të lidhen në një rrjet Docker të jashtëm, me emrin zk-net.

Cluster Node Monitoring

Qëllimi i këtij task-u është krijimi i një **cluster-i me të paktën 3 node** të Zookeeper dhe ndërtimi i një **sistemi monitorimi** për të ndjekur statusin e çdo node në kohë reale. Monitorimi duhet të përfshijë metrika të rëndësishme për gjendjen e node-ve si dhe një mënyrë për të menaxhuar **node failures** në mënyrë të qartë dhe të log-uar.

Arkitektura dhe Zgjidhja e Ndërtuar

Zgjidhja e implementuar përbëhet nga **tre komponentë kryesorë**:

1. **ZKClusterMonitor.java**
   Ky është komponenti kryesor që monitoron çdo node në cluster. Për çdo host, sistemi:
   - Lidhët përmes një socket-i TCP.
   - Dërgon komandën "stat" (një **four-letter-word command** e Zookeeper).
   - Lexon përgjigjen për të nxjerrë metrikat kryesore.

Metrikat që monitorohen për çdo node:

- **isAlive** – a është node aktiv apo jo.
- **mode** – roli aktual i node-it në cluster (p.sh. LEADER, FOLLOWER).
- **connections** – numri i lidhjeve aktive.
- **latency** – koha e përgjigjes së node-it në ms.
- **lastChecked** – koha kur node-i u kontrollua për herë të fundit.

Informacioni ruhet në një strukturë ConcurrentHashMap që përditësohet çdo **10 sekonda** përmes një ScheduledExecutorService.

2. **ZKConnectionTester.java**
   Ky klasë përdoret për të testuar numrin e lidhjeve që një node mund të përballojë. Krijohen 100 lidhje të reja me cluster-in përmes API-së ZooKeeper, duke matur sjelljen e cluster-it nën ngarkesë. Pas një minute, të gjitha lidhjet mbyllen.

3. **ZKDashboard.java**
   Një **web dashboard** i lehtë, i ndërtuar me HttpServer, që shfaq statusin aktual të çdo node në formë **HTML table**. Përditësimi bëhet çdo 2 sekonda dhe tabela përfshin:
   - Host-i
   - Statusi (ALIVE / DOWN)
   - Mode (LEADER / FOLLOWER / etc.)

- Numri i lidhjeve
- Vonesa
- Koha e fundit e kontrollit

Ky dashboard ofron një **pamje të qartë vizuale** të gjendjes së cluster-it dhe është shumë i dobishëm për **troubleshooting** dhe **menaxhimin e ngarkesave**.

Menaxhimi i Node Failures

Nëse një node nuk përgjigjet, sistemi:

- Shënon isAlive = false për atë host.
- Lëshon një log me nivel **ERROR** që tregon host-in problematik dhe shkakun (p.sh. socket timeout).
- Të metrikat e mëparshme për krahasim dhe auditim.

Kjo mënyrë e thjeshtë por efikase ndihmon për të identifikuar problemet në kohë reale dhe për të marrë masa proaktive ndaj prishjeve në cluster.

Përfundime

Ky sistem monitorimi është një zgjidhje **lightweight**, **pa varësi të jashtme**, por shumë efektive për ndjekjen e statusit të një **ZooKeeper cluster-i**. Ai mund të zgjerohet lehtësisht për:

- Alert-e përmes email ose webhook.
- Ruajtje historike të metrikave në një bazë të dhënash.
- Integrim me sisteme si **Prometheus** apo **Grafana** për vizualizim më të avancuar.

Projekti demonstron një kuptim të qartë të mënyrës se si funksionon një cluster Zookeeper, si dhe aftësi praktike në ndërtimin e zgjidhjeve të monitorimit të thjeshta dhe të qëndrueshme.

ZooKeeper Cluster Status

Last updated: Wed May 07 00:50:39 CEST 2025

Host	Status	Mode	Connections	Latency (ms)	Last Checked
localhost:2181	ALIVE	FOLLOWER	24	10	Wed May 07 00:50:32 CEST 2025
localhost:2184	ALIVE	LEADER	21	4	Wed May 07 00:50:32 CEST 2025
localhost:2185	ALIVE	FOLLOWER	19	5	Wed May 07 00:50:32 CEST 2025
localhost:2182	ALIVE	FOLLOWER	20	5	Wed May 07 00:50:32 CEST 2025
localhost:2183	ALIVE	FOLLOWER	21	7	Wed May 07 00:50:32 CEST 2025

Distributed Locking

Ky task implementon një **sistem të shpërndarë për menaxhimin e aksesit konkurent** ndaj burimeve të përbashkëta duke përdorur **Zookeeper** dhe librarinë **Apache Curator** për të realizuar *distributed locking*. Qëllimi është të sigurohet **mutual exclusion**, domethënë që vetëm një proces ose thread të ketë ndikim në një zonë kritike në një moment të caktuar.

Si funksionon implementimi?

Klasa kryesore ZKDistributedLock përdor InterProcessMutex nga Apache Curator për të menaxhuar një *mutex lock* përmes një node në Zookeeper (/locks/demo ose /locks/counter). Ajo mbështet:

- acquire() – për të marrë lock-un (bllokon derisa të jetë i disponueshëm).
- acquire(time, unit) – për të marrë lock-un me afat kohor.
- release() – për ta liruar lock-un nëse është marrë më parë.
- isAcquired() – për të kontrolluar nëse lock-u zotërohet nga thread-i aktiv.
- close() – për të liruar burimet dhe mbyllur klientin e Zookeeper-it.

Testimi i sjelljes me dhe pa locking

Klasa DistributedLockTest demonstron sjelljen e sistemit me dhe pa përdorimin e lock-ut duke përdorur CountDownLatch për të nisur disa thread-e në mënyrë simultane. Secili thread përpiqet të inkrementojë një AtomicInteger të përbashkët:

- **Pa lock**: nuk ka garanci për saktësinë e vlerës finale për shkak të race conditions.
- **Me lock**: përdor ZKDistributedLock për të garantuar që vetëm një thread hyn në seksionin kritik në çdo moment. Vlera e fundit e counter-it përputhet me vlerën e pritur.

Si ekzekutohet në IntelliJ:

1. Sigurohuni që një instancë Zookeeper të jetë aktive në localhost:2181,2182,2183.
2. Importoni projektin në IntelliJ si Maven/Gradle ose projekt Java.
3. Ekzekutoni klasën DistributedLockTest.java si aplikacion Java (Run → Run 'DistributedLockTest').
4. Log-et do tregojnë qartë se si ndërthuren thread-et dhe si garanton locking korrektësinë e ekzekutimit.

Si sigurohet mutual exclusion?

- InterProcessMutex siguron që vetëm një proces/thread mund të marrë lock-un për një lockPath të caktuar.
- Në testim, kur përdoret lock, çdo hyrje në seksionin kritik është e serializuar – shmangen race conditions.

- Metoda acquire(timeout) shmang bllokimet e përhershme në rast gabimesh (deadlock avoidance).
- Gabimet menaxhohen me try/catch dhe finally për të siguruar që lock-i të lirohet gjithmonë pas përdorimit.

Edge cases të mbuluara:

- **Timeout në marrjen e lock-ut** – parandalon bllokimin e përhershëm.
- **Thread që dështon pa lëshuar lock-un** – Curator merret me "session expiration" dhe e liron automatikisht.
- **Verifikimi nëse lock-i është marrë** (isAcquired) për të shmangur lirime të padrejta.
- **Mbyllje e rregullt e klientit** (client.close()) në close().

```
00:52:39.416 [main] INFO  com.zklock.DistributedLockTest - Starting Distributed Lock Test with 4 threads, 4 operations each
00:52:39.426 [main] INFO  com.zklock.DistributedLockTest - === TEST WITHOUT LOCKS ===
00:52:39.445 [main] INFO  com.zklock.DistributedLockTest - All threads started
00:52:39.445 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 starting
00:52:39.445 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 starting
00:52:39.450 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 starting
00:52:39.450 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 starting
00:52:39.451 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 1
00:52:39.455 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 2
00:52:39.456 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 1
00:52:39.457 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 1
00:52:39.459 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 1
00:52:39.460 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 3
00:52:39.462 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 2
00:52:39.463 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 4
00:52:39.463 [Thread-3] INFO  com.zklock.DistributedLockTest - Worker 3 completed all operations
00:52:39.464 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 2
00:52:39.467 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 3
00:52:39.467 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 2
00:52:39.471 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 4
00:52:39.474 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 3
00:52:39.474 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 5
00:52:39.474 [Thread-0] INFO  com.zklock.DistributedLockTest - Worker 0 completed all operations
00:52:39.475 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 4
00:52:39.475 [Thread-1] INFO  com.zklock.DistributedLockTest - Worker 1 completed all operations
00:52:39.481 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 4
00:52:39.481 [Thread-2] INFO  com.zklock.DistributedLockTest - Worker 2 completed all operations
00:52:39.481 [main] INFO  com.zklock.DistributedLockTest - All threads completed
00:52:39.481 [main] INFO  com.zklock.DistributedLockTest - Final counter value: 4 (Expected: 16)
```

```
00:52:41.482 [main] INFO  com.zklock.DistributedLockTest - === TEST WITH LOCKS ===
00:52:41.482 [main] INFO  com.zklock.DistributedLockTest - All threads started
00:52:41.482 [Thread-4] INFO  com.zklock.DistributedLockTest - Worker 0 starting
00:52:41.482 [Thread-6] INFO  com.zklock.DistributedLockTest - Worker 2 starting
00:52:41.483 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 starting
00:52:41.483 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 starting
00:52:41.617 [Thread-4] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:52:41.617 [Thread-7] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:52:41.617 [Thread-5] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:52:41.617 [Thread-6] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.570 [Thread-4] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.581 [Thread-4] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 1
00:53:08.581 [Thread-4] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.589 [Thread-4] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.592 [Thread-5] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.592 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 2
00:53:08.592 [Thread-5] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.596 [Thread-5] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.600 [Thread-7] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.612 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 3
00:53:08.612 [Thread-7] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.616 [Thread-7] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.618 [Thread-6] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.627 [Thread-6] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 4
00:53:08.627 [Thread-6] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.631 [Thread-6] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.634 [Thread-4] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.643 [Thread-4] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 5
00:53:08.643 [Thread-4] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.648 [Thread-4] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.649 [Thread-5] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.650 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 6
00:53:08.650 [Thread-5] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.654 [Thread-5] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.656 [Thread-7] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.673 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 7
00:53:08.673 [Thread-7] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.678 [Thread-7] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.680 [Thread-6] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.703 [Thread-6] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 8
00:53:08.703 [Thread-6] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.708 [Thread-6] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.710 [Thread-4] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.720 [Thread-4] INFO  com.zklock.DistributedLockTest - Worker 0 incremented counter to 9
00:53:08.720 [Thread-4] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.723 [Thread-4] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.724 [Thread-5] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.735 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 10
00:53:08.735 [Thread-5] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.740 [Thread-5] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.742 [Thread-7] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.751 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 11
00:53:08.751 [Thread-7] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.755 [Thread-7] INFO  com.zklock.ZKDistributedLock - Attempting to acquire lock at path: /locks/counter with timeout: 30 SECONDS
00:53:08.757 [Thread-6] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.781 [Thread-4] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.787 [Thread-5] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.797 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 incremented counter to 14
00:53:08.797 [Thread-5] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.804 [Thread-7] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.811 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 incremented counter to 15
00:53:08.811 [Thread-7] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.819 [Thread-6] INFO  com.zklock.ZKDistributedLock - Lock acquired successfully at path: /locks/counter
00:53:08.826 [Thread-6] INFO  com.zklock.DistributedLockTest - Worker 2 incremented counter to 16
00:53:08.826 [Thread-6] INFO  com.zklock.ZKDistributedLock - Releasing lock at path: /locks/counter
00:53:08.905 [Thread-4] INFO  com.zklock.DistributedLockTest - Worker 0 completed all operations
00:53:09.011 [Thread-6] INFO  com.zklock.DistributedLockTest - Worker 2 completed all operations
00:53:09.117 [Thread-7] INFO  com.zklock.DistributedLockTest - Worker 3 completed all operations
00:53:09.222 [Thread-5] INFO  com.zklock.DistributedLockTest - Worker 1 completed all operations
00:53:09.223 [main] INFO  com.zklock.DistributedLockTest - All threads completed
00:53:09.223 [main] INFO  com.zklock.DistributedLockTest - Final counter value: 16 (Expected: 16)
```

Distributed Queues

Kemi një sistem të shpërndarë që implementon radhë (queue) mbi ZooKeeper duke përdorur Apache Curator. Kjo mundëson komunikimin asinkron midis komponentëve të ndryshëm në një sistem të shpërndarë, ku shumë producentë (producers) mund të shtojnë elemente në radhë dhe shumë konsumatorë (consumers) mund t'i përpunojnë ato.

Si funksionon implementimi?

Klasa kryesore `ZKDistributedQueue<T>` ofron një radhë të shpërndarë që përdor `DistributedQueue` nga Apache Curator. Kjo klasë mbështet:

- **start()** - për të filluar radhën dhe përpunimin e mesazheve
- **put(item)** - për të shtuar një element në radhë
- **size()** - për të marrë numrin e elementeve në radhë
- **close()** - për të liruar burimet dhe mbyllur klientin e ZooKeeper

Elementët duhet të jenë të serializueshëm për t'u ruajtur në ZooKeeper dhe klasa përdor një `ObjectSerializer` për të konvertuar objektet në byte arrays dhe anasjelltas.

Si funksionon ZooKeeper për Distributed Queues?

ZooKeeper përdoret si mekanizëm për koordinimin e sistemeve të shpërndara, duke ofruar:

1. **Ruajtje të përbashkët** - Të dhënat ruhen në një strukturë hierarkike si një sistem skedarësh
2. **Sinkronizim** - Operacionet janë atomike, duke shmangur race conditions
3. **Njoftim në kohë reale** - Klientët marrin njoftime për ndryshimet në node-t që monitorojnë
4. **Sekuencë të garantuar** - Elementët e radhës procesohen në rendin e duhur

Për radhët e shpërndara, ZooKeeper krijon node-t e përkohshme sekuenciale nën path-in e specifikuar (p.sh. `/queues/demo`). Çdo element i radhës merr një numër unik sekuencial, duke siguruar rendin FIFO (First In, First Out).

Testimi i sistemit

`DistributedQueueTest` demonstron testimin e një sistemi të shpërndarë me disa prodhues dhe konsumatorë:

- **Prodhuesit (3)** - Secili prodhon një numër mesazhesh me ID unike
- **Konsumatorët (3)** - Marrin dhe përpunojnë mesazhet nga radha
- **Monitorimi** - Kontrollon që secili mesazh të përpunohet vetëm një herë (no duplicates)

Testimi përfshin:

1. Fillimin e disa konsumatorëve që dëgjojnë në radhën e përbashkët

2. Krijimin e një radhe
3. Ekzekutimin e disa producentëve paralelisht për të shtuar elemente në radhë
4. Verifikimin që të gjitha elementet përpunohen saktësisht një herë

Skenarët e testimit

Testimi me disa prodhues dhe konsumatorë që punojnë paralelisht, duke demonstruar që:

- Secili mesazh përpunohet vetëm nga një konsumator
- Nuk ka dyfishime në përpunim
- Radhët funksionojnë me ngarkesë të lartë

Mund të testohet duke:

- Ndërprerë lidhjen mes një konsumatori dhe ZooKeeper
- Monitoruar rivendosjen automatike të lidhjes
- Verifikuar që asnjë mesazh nuk humbet gjatë ndarjes së rrjetit

Simuluar duke:

- Mbyllur një konsumator gjatë përpunimit
- Verifikuar që mesazhet e tij rindahen dhe përpunohen nga konsumatorët e tjerë
- Kontrolluar për përpunim të dyfishuar pas rivendosjes

Performanca

1. **Serialization eficiente** - Implementimi i `ObjectSerializer` optimizon konvertimin e objekteve
2. **Monitorimi i madhësisë** - `AtomicInteger,queueSize` mban gjurmën e numrit të elementeve
3. **Logging të detajuar** - Për debugging dhe monitorim

Përfundim

Sistemi i implementuar ofron një mekanizëm të besueshëm për komunikim asinkron në sisteme të shpërndara, duke përdorur ZooKeeper si infrastrukturë koordinimi. Përdorimi i Apache Curator lehtëson implementimin, duke ofruar një API të lartë për operacionet komplekse të shpërndara si radhët.

```
00:58:43.385 [main] INFO  com.zkqueue.DistributedQueueTest - Starting shared queue test with thread pool consumers
00:58:43.586 [main] INFO  com.zkqueue.ZKDistributedQueue - ZKDistributedQueue initialized with path: /queues/test
00:58:43.587 [main] INFO  com.zkqueue.ZKDistributedQueue - Starting queue at path: /queues/test
00:58:43.624 [main] INFO  com.zkqueue.ZKDistributedQueue - Queue started successfully at path: /queues/test
00:58:43.624 [main] INFO  com.zkqueue.DistributedQueueTest - Shared queue consumer started
00:58:43.635 [pool-6-thread-2] INFO  com.zkqueue.DistributedQueueTest - Producer producer-1 -> Message(producer=producer-1, id=0, time=1746572323630)
00:58:43.635 [pool-6-thread-1] INFO  com.zkqueue.DistributedQueueTest - Producer producer-0 -> Message(producer=producer-0, id=0, time=1746572323630)
00:58:43.635 [pool-6-thread-3] INFO  com.zkqueue.DistributedQueueTest - Producer producer-2 -> Message(producer=producer-2, id=0, time=1746572323630)
00:58:43.635 [pool-6-thread-2] INFO  com.zkqueue.DistributedQueueTest - Producer producer-1 -> Message(producer=producer-1, id=1, time=1746572323635)
00:58:43.635 [pool-6-thread-1] INFO  com.zkqueue.DistributedQueueTest - Producer producer-0 -> Message(producer=producer-0, id=1, time=1746572323635)
00:58:43.635 [pool-6-thread-3] INFO  com.zkqueue.DistributedQueueTest - Producer producer-2 -> Message(producer=producer-2, id=1, time=1746572323635)
00:58:43.636 [main] INFO  com.zkqueue.DistributedQueueTest - Waiting for all items to be processed...
00:59:10.531 [pool-1-thread-1] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-5, id=0, time=1746570200484)
00:59:10.538 [pool-1-thread-2] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-1, id=0, time=1746570200484)
00:59:10.545 [pool-1-thread-3] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-0, id=0, time=1746570200484)
00:59:10.554 [pool-1-thread-1] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-3, id=0, time=1746570200484)
00:59:10.559 [pool-1-thread-2] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-6, id=0, time=1746570200484)
00:59:10.565 [pool-1-thread-3] INFO  com.zkqueue.DistributedQueueTest - [shared-consumer] Processing: Message(producer=producer-2, id=0, time=1746570200484)
00:59:10.565 [main] INFO  com.zkqueue.DistributedQueueTest - ✅ All items processed successfully!
00:59:10.565 [main] INFO  com.zkqueue.DistributedQueueTest - ✔ No duplicates. All items processed once.
00:59:10.565 [main] INFO  com.zkqueue.ZKDistributedQueue - Closing queue at path: /queues/test
00:59:10.566 [main] INFO  com.zkqueue.ZKDistributedQueue - Queue closed successfully at path: /queues/test
```