

Modulo Software Testing Report

Studente: Marco Calavaro

Matricola: 0295233

Introduzione e descrizione del documento

In tale documento saranno riportate le analisi, le scelte effettuate, i problemi riscontrati, le soluzioni adottate e i risultati dell'attività di testing svolta nei progetti.

La prima parte descrive il lavoro svolto nel *progetto "1+"* che consisteva nello studio di JCS e di 2 classi di test (trattato nel Capitolo I).

La seconda parte descrive il lavoro svolto nei "2" progetti BookKeeper e ZooKeeper (trattato nel Capitolo II)

Per far fronte ad alcuni problemi riscontrati nei vari progetti l'ambiente di sviluppo utilizzato varia tra progetto "1+" e progetto "2"; esso sarà descritto nei paragrafi introduttivi di ciascun capitolo, insieme alla descrizione di alcune note preliminari al lavoro svolto.

Tutti i progetti sono stati configurati per essere versionati in GitHub e con build automatica su Travis e SonarCloud. Dai nuovi aggiornamenti di Travis sono stati riscontrati dei problemi di build rispetto a versioni passate, in particolare è richiesto openjdk11 come jdk di configurazione. Inoltre la meccanica dei Credits non mi ha permesso di usufruire in maniera completa del servizio.

Si fa presente un problema riscontrato nella configurazione di SonarCloud per il progetto BookKeeper per il quale non viene riportata la coverage sul sito. Ho provato a correggere tale errore fino all'ultimo ma le configurazioni per Jacoco identiche sia per ZooKeeper che per BookKeeper, un progetto funziona perfettamente, mentre il secondo da problemi.

Di seguito i link per i vari progetti:

JCS:

- [GitHub](https://github.com/IlConteCvma/JCS): <https://github.com/IlConteCvma/JCS>
- [TravisCI](https://travis-ci.com/github/IlConteCvma/JCS): <https://travis-ci.com/github/IlConteCvma/JCS>
- [SonarCloud](https://sonarcloud.io/dashboard?id=IlConteCvma_bookkeeper): https://sonarcloud.io/dashboard?id=IlConteCvma_bookkeeper

BookKeeper:

- [GitHub](https://github.com/IlConteCvma/bookkeeper): <https://github.com/IlConteCvma/bookkeeper>
- [TravisCI](https://travis-ci.com/github/IlConteCvma/bookkeeper): <https://travis-ci.com/github/IlConteCvma/bookkeeper>
- [SonarCloud](https://sonarcloud.io/dashboard?id=IlConteCvma_bookkeeper): https://sonarcloud.io/dashboard?id=IlConteCvma_bookkeeper

ZooKeeper:

- [GitHub](https://github.com/IlConteCvma/zookeeper_2): https://github.com/IlConteCvma/zookeeper_2
- [TravisCI](https://travis-ci.com/github/IlConteCvma/zookeeper_2): https://travis-ci.com/github/IlConteCvma/zookeeper_2
- [SonarCloud](https://sonarcloud.io/dashboard?id=IlConteCvma_zookeeper_2): https://sonarcloud.io/dashboard?id=IlConteCvma_zookeeper_2

Capitolo I

Introduzione

Il primo progetto ha avuto lo scopo di sperimentare alcune tecniche di testing applicandole al progetto Apache JCS (versione 1.3), nello specifico tradurre i test da Junit3 a Junit4, convertire i test in un formato parametrico, configurare Jacoco su un progetto single module con librerie esterne.

Le classi da utilizzare selezionate tramite l'algoritmo assegnatomi sono :

- [JCSLightLoadUnitTest.java](#)
- [JCSRemovalSimpleConcurrentTest.java](#)

Il lavoro è stato svolto seguendo tali step:

- Conversione dei test e riscrittura in forma parametrica
- Creazione di un progetto maven e configurazione

L'ambiente di sviluppo :

Tale progetto è stato sviluppato su SO Windows10 tramite ide Eclipse versione 2020-12 (4.18.0) e compilato tramite Maven versione 3.8.1

Conversione dei test

La prima parte del lavoro si è incentrata sulla conversione dei test da Junit3 a Junit4 senza alterarne il funzionamento. In particolare la nuova versione di Junit ha introdotto le notazioni che vengono utilizzate per dichiarare al compilatore le classi di test e i vari metodi.

Riscrittura in forma parametrica

Completato il passaggio da una versione all'altra di Junit ho riscritto i test in forma parametrica, non alterando la logica del test stesso. Per poter far ciò le classi sono state etichettate con `@RunWith(Parameterized.class)` che indica a Junit di istanziare la classe di test passando al costruttore gli Object ottenuti tramite il metodo `getTestParameters()` etichettato con `@Parameters`, per completare il setup prima del test effettivo (logica presente anche nei test originali) viene lanciato il metodo `configure()` etichettato con `@Before`. I metodi di test (ora etichettati con `@Test`) sono stati pressoché lasciati invariati con le opportune modifiche dei parametri non più "hard-coded" ma passati tramite runner.

Il progetto maven

I test sono stati inseriti in un progetto Maven con archetipo Java (in particolare nella cartella `src/test`).

Gran parte del tempo di lavoro è stato speso nella configurazione di Jacoco per la coverage poiché il progetto Jcs è stato importato come dipendenza esterna.

La libreria è stata instrumentata tramite command line interface (CLI) di Jacoco e inserito nella cartella `/src/test/lib`.

Per poter lanciare la coverage è stato creato un profilo apposito che installa la libreria Jcs instrumentata, per risolvere uno smell generato dall'analisi di SonarCloud tale compito è stato affidato al plugin `maven-install`.

Il comando per lanciare quindi i test è: `mvn test -Pcoverage`

Ottenimento del report

Tramite il link esterno della libreria instrumentata, Jacoco è in grado di creare il solo file `jacoco.exec` che sarà posizionato da Maven nella cartella `target`. Per poter generare la pagina di report e i relativi file annessi, Jacoco ha bisogno della libreria non instrumentata.

Il report è stato quindi generato attraverso CLI passando esplicitamente lo jar di Jcs1.3 e i file generati posizionati sempre sotto `target`.

Analisi della copertura strutturale

Poiché gli unici test eseguiti nel progetto sono quelli da me creati, la copertura generata da Jacoco corrisponde esattamente a ciò che il mio test va a coprire. Di seguito saranno riportate le immagini risultanti dalla prima analisi.

Ai fini di aumentare la coverage sono state analizzate solo le classi che presentano chiamate ai metodi direttamente dai test:

- `org.apache.jcs.JCS`
- `org.apache.jcs.access.CacheAccess`

I metodi chiamati sono put e remove di CacheAccess:

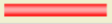











- remove presenta una coverage del 100%
- mentre la put 60% line coverage e 50% branch coverage

Analizzando con approccio white box il metodo put per provare ad alzare la coverage si è scoperto che:

- Come primo parametro non è stato passato il caso con Object key pari a null, per poter implementare tale caso bisogna modificare il test.
- Similmente anche per il secondo parametro Object val non è stato coperto il caso in cui è null, bisognerebbe modificare i test per coprirlo.
- Non si può entrare dentro il branch di tipo catch poiché l'eccezione non dipende dai parametri in input ma dal funzionamento dei metodi di un altro oggetto.

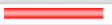





































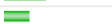





















JCS

Source file "org/apache/jcs/JCS.java" was not found during generation of report.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getInstance(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
ensureCacheManager()		76%		75%	1	3	2	7	0	1
getInstance(String)		100%		n/a	0	1	0	2	0	1
JCS(CompositeCache)		100%		n/a	0	1	0	2	0	1
setConfigFilename(String)		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	12 of 40	70%	1 of 4	75%	2	8	4	16	1	6

CacheAccess

Source file "org/apache/jcs/access/CacheAccess.java" was not found during generation of report.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
freeMemoryElements(int)		0%		n/a	1	1	8	8	1	1
resetElementAttributes(Object, IElementAttributes)		0%		0%	2	2	5	5	1	1
putSafe(Object, Object)		0%		0%	2	2	4	4	1	1
ensureCacheManager()		0%		0%	3	3	5	5	1	1
put(Object, Object, IElementAttributes)		60%		50%	2	3	4	11	0	1
getElementAttributes(Object)		0%		n/a	1	1	6	6	1	1
destroy()		0%		n/a	1	1	5	5	1	1
defineRegion(String, ICompositeCacheAttributes, IElementAttributes)		0%		n/a	1	1	2	2	1	1
defineRegion(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
getAccess(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
defineRegion(String)		0%		n/a	1	1	2	2	1	1
getAccess(String)		0%		n/a	1	1	2	2	1	1
destroy(Object)		0%		n/a	1	1	2	2	1	1
clear()		45%		n/a	0	1	2	5	0	1
getCacheElement(Object)		0%		n/a	1	1	1	1	1	1
resetElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
setDefaultElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
getElementAttributes()		0%		n/a	1	1	1	1	1	1
getDefaultElementAttributes()		0%		n/a	1	1	1	1	1	1
getStatistics()		0%		n/a	1	1	1	1	1	1
dispose()		0%		n/a	1	1	2	2	1	1
remove()		0%		n/a	1	1	2	2	1	1
static {...}		90%		50%	1	2	0	1	0	1
get(Object)		100%		100%	0	2	0	2	0	1
put(Object, Object)		100%		n/a	0	1	0	2	0	1
remove(Object)		100%		n/a	0	1	0	2	0	1
CacheAccess(CompositeCache)		100%		n/a	0	1	0	3	0	1
setCacheAttributes(ICompositeCacheAttributes)		100%		n/a	0	1	0	2	0	1
getStats()		100%		n/a	0	1	0	1	0	1
getCacheAttributes()		100%		n/a	0	1	0	1	0	1
Total	241 of 327	26%	11 of 16	31%	27	38	63	87	20	30

Capitolo II

Introduzione

Progetti Trattati:

- BookKeeper versione 4.11.1
- ZooKeeper versione 3.5.9

Similmente al progetto affrontato nel Capitolo 1, il lavoro svolto per i due progetti ha avuto lo scopo di sperimentare alcune tecniche di testing, ma in questo caso i test sono stati generati partendo da zero.

La scelta delle classi da testare è stata guidata dall'analisi svolta per il Modulo del professor Falessi per il progetto ZooKeeper, mentre è stata tenuta in considerazione inferiormente per BookKeeper perché per tale modulo è stata utilizzata la versione 4.5.2 (risalente al 2017) poiché i ticket reperibili su Jira si fermano alla data del 17/Oct/2017 come riportato nel ticket BOOKKEEPER-1107. Per tale Modulo è stato scelto di lavorare con la versione *Stable* (4.11.1)

Da evidenziare anche la scelta della versione di ZooKeeper che inizialmente è stata quella di prendere l'ultima release 3.7.0 ma a causa di problemi legati alla configurazione dei vari moduli maven. Nello specifico il testing viene eseguito attraverso Jupiter (org.junit.jupiter.api) anziché Junit4. Per non investire eccessivo tempo nella risoluzione di alcuni problemi nel cambio di dipendenze della libreria di test si è preferito ritornare alla versione 3.5.9 che sfruttava ancora Junit4 per i test.

Scelta delle classi

Si sono cercate classi rilevanti per l'applicazione dei concetti affrontati nel Modulo di Testing tenendo conto maggiormente di quelle misurate come buggy nelle ultime release e che presentassero valori alti per le metriche di LOC, LOC_touched, NR e, con impatto inferiore, le altre metriche.

La scelta delle classi è stata in seguito condizionata da:

- Documentazione reperibile sulla classi e sui metodi che essa esporta.
- Commenti all'interno della classe.
- Istanziamento chiara, deducibile o dalla documentazione o dal codice, e eventuali classi di test di supporto.
- "Validi" metodi da testare

Con "validi" si intende metodi con parametri non eccessivamente complessi sui quali si è potuto applicare category partition in modo chiaro, sono stati quindi scartati quei metodi che richiedevano una elevata conoscenza sugli oggetti "complessi" del sistema. (Questo perché molto spesso non è stato possibile trovare una documentazione chiara ed esaustiva di tali classi)

Nella filosofia di ricerca di tali metodi "validi" si è tenuto conto anche del tipo di dato ritornato o dell'eccezioni che tale metodo ritornava.

L'ambiente di sviluppo :

La compilazione del progetto BookKeeper tramite Maven su Windows10 riscontrava dei problemi, nello specifico il modulo circe-checksum. Ho analizzato la configurazione del pom di tale modulo e ho notato che vi sono specifici profili per i sistemi operativi Unix-like ma non per Windows sul repository di tale progetto è stata proposta una soluzione (in attesa di approvazione) ma, anche adottando le modifiche proposte, sono rimasti dei problemi legati al compilatore g++.(Non è facile reperire la versione corretta su Windows).

Poiché tale progetto da documentazione è previsto che giri su Linux ho deciso di cambiare ambiente di sviluppo per i due progetti di questo Capitolo passando a Linux Ubuntu-Mate 20.04 LTS, sfruttando come IDE IntelliJ e compilando sempre tramite maven versione 3.8.1

Category Partition

Di seguito saranno trattate le scelte prese nel definire i casi di test per i due progetti.

BookKeeper

Classi analizzate: EntryLogger.java , BookKeeper.java

EntryLogger

Descrizione della classe:

La classe ha lo scopo di gestire la scrittura delle entries di BookKeeper. In particolare le entries vengono scritte sul common log e i puntatori vengono memorizzati sulla LedgerCache per averne un accesso rapido. Il metodo che è stato testato è

```
public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry) throws IOException
```

Descrizione del metodo:

Il metodo serve per andare a leggere sul log la entry corrispondente ad entryId per il ledger definito da ledgerId.

- LedgerId è un long che rappresenta l'identificativo di un ledger di cui si sta cercando informazioni nel log. Da documentazione si evince che tale valore deve essere \geq di zero per essere valido.
- EntryId long che rappresenta l'identificativo di un entry che dovrebbe trovarsi nel log. Anche qui da documentazione si evince che entryId deve essere \geq di zero per essere valido.
- Location indica il puntatore alla posizione dell'entry nell'entry log.
- ValidateEntry è un booleano che se true effettua il controllo della validità dell'entry passata e controlla se effettivamente esiste, se false ignora il controllo e legge i byte nella posizione fissata.

Descrizione dei test:

❖ (Classe MyEntryLoggerTest)

Per testare la funzionalità dell' internalReadEntry sono stati creati due metodi di test che lanciano la funzione privata testInternalReadEntry che si aspetta un ledgerId e entryId usati per generare un entry nel log file (N.B. non sono i valori usati per la chiamata dell'internalReadEntry anche se omonimi la chiamata al metodo testato viene effettuata con i valori definiti dal costruttore).

In particolare i due test sono:

- testInternalReadEntryGoodPosition prima della chiamata al metodo viene generata una entry nella posizione corretta (stessi valori di ledgerId e entryId passati nel costruttore).
- testInternalReadEntryWrongPosition test simile ma la entry generata si trova in una posizione differente rispetto a quella ricercata.

Tramite runner Parametrized vengono inizializzati diversi valori per ledgerId, entryId e validateEntry ma non per location che viene istanziata dal metodo addEntry. Non ho effettuato il domain partitioning su questo valore poiché la logica pensata per il test richiede che tale posizione sia valida. Si potrebbe modificare il test sapendo (dalla documentazione) che ogni qual volta la position non rappresenta un valore nel logFile verrà sollevata una IndexOutOfBoundsException ma per far ciò si introdurrebbe una complicazione del metodo di test per eseguire un solo test case in più.

I valori scelti dal domain partitioning per eseguire i casi di test sono: (ledgerId, entryId, validateEntry) {0L,0L,true}, {1L,0L,true}, {-1L,0L,true}, {0L,-1L,true}, sono stati disabilitati i test con valori false poiché per il funzionamento di internalReadEntry, nel momento in cui non viene effettuato alcun controllo sull'entry cercata la funzione si comporta come una semplice read che restituisce i byte specificati a partire da position e come aspettato i test falliscono poiché gli assert di controllo sui valori di ritorno della funzione non trovano i valori aspettati

❖ (Classe MyEntryLoggerMultiLedgersTest)

Questa classe testa comunque il metodo internalReadEntry ma cambia la logica che vi è dietro. In questo caso non ho voluto testare la classe partendo dallo studio dei parametri ma dal suo funzionamento, andando a verificare più letture consecutive su differenti ledger. Inoltre analizzando i parametri di configurazione ho notato che è possibile alterare la creazione dei log per ledger (parametro setEntryLogPerLedgerEnabled). In base al valore settato l'internalReadEntry deve effettuare letture in posizioni diverse. Si va quindi a testare il corretto comportamento sotto tale configurazione.

BookKeeper

Descrizione della classe:

Classe che rappresenta il BookKeeper client espone 4 possibili operazioni in modo sincrono o asincrono (Start, write, read, close di un ledger).

La scelta di tale classe è stata guidata dal fatto che è una delle classi con maggiore descrizione del funzionamento dei metodi, facilitando quindi il lavoro di comprensione del contesto di applicazione e operabilità della classe. Portando ad una analisi del dominio più semplice e chiara rispetto alla precedente classe analizzata.

Essendo il fulcro del funzionamento dell'applicativo, il suo setUp richiedeva comunque conoscenze avanzate del sistema, perciò ho sfruttato delle classi di test presenti nel progetto per ottenere un ambiente di test corretto.

Per studiare un meccanismo da me poco utilizzato in Java ho scelto di analizzare e testare funzioni asincrone. Il metodo testato è:

```
public void asyncCreateLedger(final int ensSize, final int writeQuorumSize, final int
ackQuorumSize, final DigestType digestType, final byte[] passwd, final CreateCallback
cb, final Object ctx, final Map<String, byte[]> customMetadata)
```

Descrizione del metodo:

Il metodo viene utilizzato per creare in modo asincrono un nuovo ledger, a differenza della create sincrona in questo metodo è possibile specificare la dimensione del writeQuorum e ackQuorum.

- EnsSize rappresenta la dimensione dell'ensemble cioè del numero di nodi in cui il ledger deve essere memorizzato
- WriteQuorumSize è il numero di nodi sul quale una entry salvata (max replication)
- AckQuorumSize è il numero di nodi sul quale una entry deve essere riconosciuta (min replication)
- DigestType definisce la metodologia di creazione del digest (crc32 o mac)
- Passwd la password del ledger
- Cb rappresenta l'elemento di callback
- ctx è un oggetto opzionale di controllo
- CustomMetadata è un oggetto di configurazione che permette di inizializzare il ledger con nuovi metadata

Dalla documentazione inoltre si evince che $\text{ensSize} \geq \text{writeQuorumSize} \geq \text{ackQuorumSize}$

Descrizione dei test:

❖ (Classe MyAsyncCreateLedgerTest)

Tale classe di test verifica il corretto comportamento del metodo su descritto, in particolare tramite runner Parametrized viene istanziata la test class facendo variare i valori di ensSize, writeQuorumSize, ackQuorumSize.

I vari test case cercano di coprire i possibili risultati che si evincono dall'analisi del tipo di dato e dalla documentazione (I singoli valori sono stati scelti in modo randomico rispettando i possibili casi logici dati dalla combinazione dei tre parametri).

Test case: {7,6,4}, {7,2,4}, {1,2,3}, {0,0,0}, {2,0,0}, {2,-1,-3}, {-1,3,2}

Il test effettua il controllo della riuscita dell'operazione aspettando il completamento della callback, l'attesa viene effettuata tramite wait sull'oggetto ctx passato alla funzione.

Tale controllo consiste nel verificare che i metadata ritornati nel ledgerHandle siano corretti o eventuali codici di errore sollevati siano tra quelli attesi, in particolare ci si aspetta di ricevere IllegalArgumentException con messaggio :

- "Illegal Capacity" se ensemble size negative.
- "Write quorum must be larger than ack quorum".

Eventuali codici di errore diversi comporterebbero il fallimento del test.

ZooKeeper

Classi analizzate: ZooKeeper.java , QuorumPeer

Per la creazione di test per questo progetto ho deciso di iniziare con un approccio leggermente diverso, cercando di redigere un test esclusivamente attraverso la documentazione.

Nel capitolo "[The ZooKeeper Data Model](#)" vengono descritte le regole di assegnazione dei path ai vari nodi. Un path deve essere sempre assoluto e qualsiasi carattere Unicode può essere utilizzato per la sua rappresentazione, ad eccezione di diversi casi legati al fatto che alcuni caratteri possono creare malfunzionamenti del sistema. (Per brevità riportati nella descrizione della classe di test sviluppata)

Ci si aspetta quindi che nei metodi che operano sui nodi vi siano gli opportuni controlli e che in caso di path non conforme a tali regole sia restituito un errore. Ho quindi selezionato i possibili candidati al test e tra questi vi è sicuramente la creazione di un nodo, da qui la scelta della prima classe under test.

Per completezza di analisi ho poi analizzato il metodo "create" con domain partitioning e sviluppato un'altra test class ad hoc che, pur essendo simile, ha una logica di fondo differente.

ZooKeeper

Metodi analizzati: create e delete

Descrizione della classe:

Questa classe è la main class del client dell'omonimo progetto. Permette di usufruire dei vari meccanismi offerti da essa ed è quindi uno dei primi oggetti istanziati.

Per poter operare su questa classe ho sfruttato la classe di testing "ClientBase" che si occupa di istanziare correttamente un client ZooKeeper. (Per poter funzionare tale classe si aspetta di trovare la cartella target/surefire, viene creata automaticamente da maven durante la fase di test grazie ad una opportuna configurazione del profilo di coverage).

Classe di test da documentazione: MyPathValidationTest

Il test si occupa di verificare che il metodo ZooKeeper.create(...) restituisca una IllegalArgumentException quando viene specificato un path non valido, è stato inoltre creato un secondo metodo che testa stringhe valide poiché alcune regole non escludevano caratteri, ma specificavano un comportamento di essi, in questo modo si è verificato sia il contesto valido che quello non valido. (Tali valori vengono passati dal runner Parametrized).

Il dominio che si evince dalla documentazione è abbastanza ampio (alcune regole escludono un insieme di caratteri), ho quindi scelto di prendere un sottoinsieme significativo, almeno un caso di test per regola.

Poiché l'intento di questo test non era verificare il corretto funzionamento della create, ma solo accertarsi che fossero rispettate le regole di validazione del path, i vari test case fanno variare il solo parametro path della create mentre gli altri valori sono impostati in modo standard.

Di seguito invece viene trattato lo studio whitebox del metodo.

```
public String create(final String path, byte data[], List<ACL> acl, CreateMode createMode)
```

Descrizione del metodo:

Tale metodo si occupa della creazione di un nodo nel path specificato, se già esiste un nodo viene lanciato un errore del tipo KeeperException.NodeExists, se non esiste il nodo padre viene sollevato KeeperException.NoNode, se si cerca di creare un nodo figlio partendo da un padre effimero viene lanciato KeeperException.NoChildrenForEphemerals.

- Path stringa rappresentativa del path del nuovo nodo
- Data sono i dati iniziali del nodo, max data pari a 1 MB (1,048,576 bytes)
- Acl access control list del nodo
- CreateMode definisce la modalità di creazione del nodo

Descrizione dei test:

❖ (Classe MyCreateTest)

Tale classe completa il lavoro sulla funzione create ponendola nei vari casi di errore.

È stato disabilitato il test con overflow dei dati da inserire poiché non viene restituito un valore di errore aspettato.

L'analisi del secondo metodo che è stato scelto per effettuare un altro studio di funzione asincrona.


```
delete(final String path, int version, VoidCallback cb, Object ctx)
```

Descrizione del metodo:

Rappresenta la versione asincrona della delete, ha quindi il compito di eliminare il nodo al path dato.

- Path il path del nodo da eliminare
- Version il valore della versione del nodo aspettata
- Cb callback che viene eseguita al completamento
- Ctx oggetto opzionale usato per il controllo della callback

Descrizione dei test:

(Classe MyDeleteAsyncTest)

Tramite il runner parametrized vengono definiti il path e la versione del nodo da eliminare.

Nella fase di setUp viene creato un nodo padre e un nodo figlio, i vari casi di test stimolano i possibili errori definiti dalla documentazione per la funzione delete. Ci si aspetta quindi:

- Error 0 : non vi sono stati errori si controlla che il path nella call back sia corretto
- KeeperException.BADVERSION : nel caso in cui la versione sia sbagliata
- KeeperException.NONODE : nel caso non sia stato trovato un nodo al path specificato
- KeeperException.NOTEMPTY : se si cerca di eliminare un nodo padre

Parametri: (String path , int version)

={"/test/sub",-1}, {"test/sub",1}, {"fake",-1}, {"test",-1}

QuorumPeer

Metodi analizzati: processReconfig

Descrizione della classe:

Classe centrale che definisce il protocollo del quorum, parte del processo di elezione di un nuovo leader di ZooKeeper. Tale classe può trovarsi in diversi stati, ciascuno di essi rappresenta una fase del processo di elezione. Lo stato per definizione è unico.

Per configurare un ambiente minimo al funzionamento di questa classe, piuttosto che instaurare un vero e proprio processo di elezione, ho preferito istanziare gli stab necessari al funzionamento della classe e del metodo considerato. La generazione di essi è stata fatta tramite le funzionalità della libreria Mockito.

```
public boolean processReconfig(QuorumVerifier qv, Long suggestedLeaderId, Long zxid, boolean restartLE)
```

Descrizione del metodo:

Tale metodo permette di riconfigurare il processo di elezione ed eventualmente restartare l'elezione.

- QuorumVerifier: validatore per il quorum
- SuggestedLeaderId: valore del leader
- zxid: zookeeper transaction id
- restartLE: booleano che se true riavvia l'operazione di elezione del leader

Descrizione dei test:

(Classe MyQuorumTest)

Il test ha lo scopo di verificare il corretto comportamento del metodo al variare dei suoi parametri di input. Nel metodo etichettato con before viene effettuato un setup della classe under test per impostare un semplice quorum verifier mocked che restituisce come versione 0L e un currentVote (corrisponde al voto effettuato dal peer fa parte del protocollo di elezione di ZooKeeper) di valori id = 0L e zxid = 0L.

Il primo termine passato dal runner Parametrized è isReconfigurable che viene utilizzato per permettere la riconfigurazione del QuorumPeer, se impostato a false il metodo processReconfig non effettuerà nessuna modifica e restituirà immediatamente false. Per il funzionamento del metodo deve aver valore true per questo i casi di test successivi sono impostati a true.

Essendo il metodo testato in un ambiente "isolato" che non prevede un setup del sistema nel suo complesso il valore di zxid non impatta in maniera evidente sul comportamento del metodo. Sempre legato a tali motivazioni se impostato a true il valore di restartLE vengono sollevate delle eccezioni poiché il sistema non è realmente in "running".

I test case eseguiti sono (isReconfigurable, mockQuorumVer, suggestedLeaderId, zxid, restartLE):

{false,0L,0L,0L,false}, {true,0L,0L,0L,false}, {true,1L,0L,0L,false}, {true,1L,0L,1L,false}, {true,1L,1L,1L,false}, {true,1L,-1L,-1L,false}, {true,1L,null,null,false}

Adeguatezza dei casi di test

In questa parte sarà analizzata l'adeguatezza dei casi di test basata su statement e branch coverage ricavata dall'analisi di Jacoco, le relative analisi che tali valori hanno comportato e eventuali miglioramenti.

BookKeeper

❖ I risultati per i metodi della classe EntryLogger sono:

InternalReadEntry presenta:

- statement coverage pari al 55%
- branch coverage pari al 50%

Al fine di migliorare tali valori sono andato ad analizzare i risultati di Jacoco per poter capire come raggiungere più branch. Considerando la branch a riga 841 che effettua un controllo sul valore di ritorno dei metodi readFromLogChannel e readEntrySize e non avendo accesso diretto al funzionamento di tali metodi (privati) non è stato possibile migliorare la branch coverage. Similmente per lo statement poiché non si riesce ad accedere al catch che si aspetta EntryLookupException.MissingEntryException alla riga 832 che dipende da errori che si possono verificare durante l'esecuzione di readEntrySize.

❖ I risultati per i metodi della classe BookKeeper sono:

asyncCreateLedger presenta:

- statement coverage pari al 85%
- branch coverage pari al 75%

Per poter raggiungere la branch mancante (riga 833) è necessario che prima della create il client BookKeeper venga chiuso.

Per far ciò si è introdotto un if nel test che chiude il client prima della chiamata al metodo testato. Il valore booleano che permette tale chiusura è stato aggiunto al costruttore, i precedenti casi di test hanno tale valore impostato a false, mentre è stato aggiunto un caso di test con tale valore a true. Per poter controllare l'effettivo errore ho modificato riga 152 della mia classe di test che effettuava il controllo su un singolo tipo di errore (inizialmente :

```
assertEquals(BKException.BKNotEnoughBookiesException.class,bkException.getClass()))
```

Con le modifiche apportate e il nuovo caso di test si raggiunge statement e branch coverage del 100%

ZooKeeper

❖ I risultati per la classe ZooKeeper sono:

Create presenta:

- statement coverage del 83%
- branch coverage del 60%

Analizzando il report di Jacoco ho notato che non avendo inserito la possibilità di definire l'acl non viene percorsa la branch che controlla se essa ha dimensione 0 e non è null. Per correggere ho quindi introdotto un nuovo termine passato tramite runner parametrized che definisce acl e aggiunto un caso con acl vuota. Approfondendo l'analisi inoltre in caso di acl null ho trovato che tale funzione deve restituire un errore del tipo MARSHALLINGERROR, ho quindi aggiunto tale errore come valore aspettato nel test nell' if a riga 86. Apportando queste prime modifiche ho raggiunto una statement coverage del 88% e branch coverage del 80%.

Non sono riuscito a migliorare ulteriormente tali valori poiché la branch mancante non è percorribile, essa infatti dipende dal funzionamento della classe ClientCnxn.

Delete presenta:

statement coverage del 93%

coverage del 50%

Grazie all'analisi della coverage ho potuto trovare un caso di test che non avevo considerato, cioè la delete sul nodo padre ("").

Aggiungendo quindi un caso di test che effettuasse l'operazione su tale nodo ho raggiunto il 100% di statement e branch coverage.

❖ I risultati per la classe QuorumPeer sono:

ProcessReconfig presenta:

- statement coverage del 75%
- coverage del 46%

Tra i vari metodi analizzati sicuramente questo è quello con il numero maggiore di branch. Per poter percorrere molte di esse è necessario che sia istanziato un ambiente completo molto simile a quello operativo dell'applicativo. Ciò mi ha mostrato come con il semplice variare dei parametri di input molto spesso non sia possibile attraversare tutti i branch di una funzione, ed è quindi necessario adottare un approccio diverso per effettuare il test.

In questo caso per poter migliorare i valori di coverage è necessario cambiare la natura del test, pensato per essere applicato "offline", e trasformarlo in uno più complesso che effettui un setup anche minimo dell'ambiente. Per non cambiare la natura del test ho preferito mantenere tale classe di test inalterata non potendo quindi raggiungere dei valori migliori.

Mutation Testing

Intro

Per effettuare le mutazioni è stata utilizzata la libreria pitest versione 1.5.1, non la più recente per conformità alla versione usata durante il corso per fare delle prove, la nuova versione presenterebbe alcune modifiche ai parametri di configurazione.

La configurazione di tale libreria è risultata più semplice rispetto agli altri strumenti, questo perché il plugin non va in conflitto con la configurazione dei pom originali dei progetti. A differenza di Jacoco che ha presentato diverse difficoltà.

Per il progetto ZooKeeper si è presentata una complicazione nota, legata alle classi di test native utilizzate per lo sviluppo di quelle nuove, che richiedevano la presenza della directory target/surefire per eseguire i test. Non potendo configurare come fatto per Jacoco ho preferito modificare le classi di interesse seguendo il commit legato al ticket [ZOOKEEPER-3571](#).

Per ragioni di velocità ho deciso di impostare il numero massimo di mutazioni a 30 e nel progetto di BookKeeper che presenta una maggiore lentezza impostare un numero di thread a 3 (di default a 1).

Per il contesto in cui tali test sono eseguiti ho preferito lasciare gli altri valori di pitest alla loro configurazione standard.

Analisi dei risultati

In questa parte analizzerò i risultati di pitest sulle classi analizzate

BookKeeper

I risultati riportati per tale progetto sono:

- La classe BookKeeper.java presenta una line coverage del 32% e una mutation coverage 15%.

Confrontando la line coverage con quella di Jacoco si può notare che i risultati sull'intera classe sono leggermente differenti (Jacoco riporta una coverage del 30% 97 righe coperte su 307)

Analizzando le mutazioni per il metodo testato la variazione della regola a riga 828 non viene uccisa poiché il test si aspetta quel tipo di messaggio di errore ma non effettua il controllo che la causa sia corretta.

Andrebbe quindi verificato anche nel test che solo nel caso in cui writeQuorumSize < ackQuorumSize il messaggio di errore è quello aspettato. Questa modifica porta tale mutazione nello stato killed. (Ai fini della discussione tale modifica non è stata apportata per far sì che il report di pitest la evidenzi). Le mutazioni rimanenti vanno nello stato di timeout e non evidenziano errori nel codice di test.

- La classe EntryLogger.java presenta una line coverage del 34% e una mutation coverage 17%.

Anche in questo caso confrontando la line coverage con quella di Jacoco si è ottenuto un risultato leggermente più alto. (Jacoco riporta una coverage del 32% 123 righe su 366)

Per quanto riguarda il metodo analizzato sono state catturate tutte le mutazioni.

ZooKeeper

I risultati riportati per tale progetto sono:

- La classe ZooKeeper.java presenta una line coverage del 17% e una mutation coverage 9%.

Mi aspettavo che l'analisi di pitest rilevasse valori abbastanza bassi poiché tale classe esporta molte funzionalità di cui solo 2 testate.

Per il metodo create non è stato possibile rilevare tutte le mutazioni perché alcuni branch non vengono percorsi dal test.

Per il metodo delete la mutazione a riga 1955 che rimuove la chiamata a PathUtils non è stato possibile rilevare tale mutazione, questo non dipende dal test ma dal funzionamento del metodo che se non viene validato il path effettua comunque l'operazione (comportamento simile alla create).

- La classe QuorumPeer.java presenta una line coverage del 16% e una mutation coverage 9%.

Essendo anche questa una classe molto grande in termini di LOC presenta dei valori bassi di copertura.

Il metodo analizzato presenta delle mutazioni non rilevate, molte delle quali dipendono da branch non percorsi e analizzate nella sezione test.

Conclusioni

Nel complesso il lavoro è stato abbastanza complicato e oneroso in termini di tempo. Gran parte di esso è stato speso per risolvere problemi di configurazione dei vari pom e scelta delle classi da analizzare.

La scelta delle classi da analizzare è stata guidata dal lavoro compiuto nel Deliverable 2 del professor Falessi, dalla documentazione ricavabile e dalla presenza di metodi utili ai fini di analisi richiesti nel progetto.

Non sempre è stato possibile ottenere una buona descrizione delle classi dalla documentazione e il loro comportamento è stato dedotto dal codice, ma trattandosi di oggetti complessi non sempre sono riusciti ad ottenere il risultato desiderato.

Tale lavoro è stato inoltre influenzato dalla natura dei due progetti Apache. In particolare ZooKeeper è un progetto molto stabile, con una buona documentazione, ma che presenta numerose classi complesse e che richiedono una giusta configurazione dell'ambiente per essere testate. Mentre BookKeeper è un progetto più "recente" che presenta una maggiore instabilità e quindi un numero maggiore di classi selezionabili, ma fornisce una scarsa documentazione soprattutto nel momento in cui si cercano informazioni "under the hood". Anche per tale progetto è richiesta una conoscenza minima sull'ambiente per poter eseguire i test in modo consono.

```

815.
816.     public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry)
817.         throws IOException, Bookie.NoEntryException {
818.         long entryLogId = logIdForOffset(location);
819.         long pos = posForOffset(location);
820.
821.
822.         BufferedReadChannel fc = null;
823.         int entrySize = -1;
824.         try {
825.             fc = getFCForEntryInternal(ledgerId, entryId, entryLogId, pos);
826.
827.             ByteBuf sizeBuff = readEntrySize(ledgerId, entryId, entryLogId, pos, fc);
828.             entrySize = sizeBuff.getInt(0);
829.             if (validateEntry) {
830.                 validateEntry(ledgerId, entryId, entryLogId, pos, sizeBuff);
831.             }
832.             } catch (EntryLookupException.MissingEntryException entryLookupError) {
833.                 throw new Bookie.NoEntryException("Short read from entrylog " + entryLogId,
834.                     ledgerId, entryId);
835.             } catch (EntryLookupException e) {
836.                 throw new IOException(e.toString());
837.             }
838.
839.             ByteBuf data = allocator.buffer(entrySize, entrySize);
840.             int rc = readFromLogChannel(entryLogId, fc, data, pos);
841.             if (rc != entrySize) {
842.                 // Note that throwing NoEntryException here instead of IOException is not
843.                 // without risk. If all bookies in a quorum throw this same exception
844.                 // the client will assume that it has reached the end of the ledger.
845.                 // However, this may not be the case, as a very specific error condition
846.                 // could have occurred, where the length of the entry was corrupted on all
847.                 // replicas. However, the chance of this happening is very very low, so
848.                 // returning NoEntryException is mostly safe.
849.                 data.release();
850.                 throw new Bookie.NoEntryException("Short read for " + ledgerId + "@"
851.                     + entryId + " in " + entryLogId + "@"
852.                     + pos + "(" + rc + "!=" + entrySize + ")", ledgerId, entryId);
853.             }
854.             data.writerIndex(entrySize);
855.
856.             return data;
857.         }
858.
859.
860.
861.
862.     public void asyncCreateLedger(final int ensSize, final int writeQuorumSize, final int ackQuorumSize,
863.         final DigestType digestType, final byte[] passwd,
864.         final CreateCallback cb, final Object ctx, final Map<String, byte[]> customMetadata)
865.     {
866.         if (writeQuorumSize < ackQuorumSize) {
867.             throw new IllegalArgumentException("Write quorum must be larger than ack quorum");
868.         }
869.         closeLock.readLock().lock();
870.         try {
871.             if (closed) {
872.                 cb.createComplete(BKException.Code.ClientClosedException, null, ctx);
873.                 return;
874.             }
875.             new LedgerCreateOp(BookKeeper.this, ensSize, writeQuorumSize,
876.                 ackQuorumSize, digestType, passwd, cb, ctx,
877.                 customMetadata, WriteFlag.NONE, clientStats)
878.                 .initiate();
879.             } finally {
880.                 closeLock.readLock().unlock();
881.             }
882.         }
883.     }
884.
885.
886.

```

Risultati metodi BookKeeper tramite Jacoco

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
2	33% <div><div></div><div>226/687</div></div>	16% <div><div></div><div>45/283</div></div>

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
org.apache.bookkeeper.bookie	1	34% <div><div></div><div>126/372</div></div>	17% <div><div></div><div>26/156</div></div>
org.apache.bookkeeper.client	1	32% <div><div></div><div>100/315</div></div>	15% <div><div></div><div>19/127</div></div>

```
825     public void asyncCreateLedger(final int ensSize, final int writeQuorumSize, final int ackQuorumSize,
826                                   final DigestType digestType, final byte[] passwd,
827                                   final CreateCallback cb, final Object ctx, final Map<String, byte[]> customMetadata) {
828 2      if (writeQuorumSize < ackQuorumSize) {
829          throw new IllegalArgumentException("Write quorum must be larger than ack quorum");
830      }
831 1      closeLock.readLock().lock();
832      try {
833 1          if (closed) {
834 1              cb.createComplete(BKException.Code.ClientClosedException, null, ctx);
835              return;
836          }
837          new LedgerCreateOp(BookKeeper.this, ensSize, writeQuorumSize,
838                             ackQuorumSize, digestType, passwd, cb, ctx,
839                             customMetadata, WriteFlag.NONE, clientStats)
840 1          .initiate();
841      } finally {
842 1          closeLock.readLock().unlock();
843      }
844  }
845
816     public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry)
817         throws IOException, Bookie.NoEntryException {
818         long entryLogId = logIdForOffset(location);
819         long pos = posForOffset(location);
820
821         BufferedReadChannel fc = null;
822         int entrySize = -1;
823         try {
824             fc = getFCForEntryInternal(ledgerId, entryId, entryLogId, pos);
825             ByteBuf sizeBuff = readEntrySize(ledgerId, entryId, entryLogId, pos, fc);
826             entrySize = sizeBuff.getInt(0);
827             if (validateEntry) {
828 1                 validateEntry(ledgerId, entryId, entryLogId, pos, sizeBuff);
829             }
830         } catch (EntryLookupException.MissingEntryException entryLookupError) {
831             throw new Bookie.NoEntryException("Short read from entrylog " + entryLogId,
832                                               ledgerId, entryId);
833         } catch (EntryLookupException e) {
834             throw new IOException(e.toString());
835         }
836
837         ByteBuf data = allocator.buffer(entrySize, entrySize);
838         int rc = readFromLogChannel(entryLogId, fc, data, pos);
839 1         if (rc != entrySize) {
840             // Note that throwing NoEntryException here instead of IOException is not
841             // without risk. If all bookies in a quorum throw this same exception
842             // the client will assume that it has reached the end of the ledger.
843             // However, this may not be the case, as a very specific error condition
844             // could have occurred, where the length of the entry was corrupted on all
845             // replicas. However, the chance of this happening is very very low, so
846             // returning NoEntryException is mostly safe.
847             data.release();
848             throw new Bookie.NoEntryException("Short read for " + ledgerId + "@"
849                                               + entryId + " in " + entryLogId + "@"
850                                               + pos + "(" + rc + "!=" + entrySize + ")", ledgerId, entryId);
851         }
852         data.writerIndex(entrySize);
853
854 1         return data;
855     }
```

Report pitest per BookKeeper

```

1918.     public void setAcceptedEpoch(long e) throws IOException {
1919.         writeLongToFile(ACCEPTED_EPOCH_FILENAME, e);
1920.         acceptedEpoch = e;
1921.     }
1922.
1923.     public boolean processReconfig(QuorumVerifier qv, Long suggestedLeaderId, Long zxid, boolean restartLE) {
1924.         if (!isReconfigEnabled()) {
1925.             LOG.debug("Reconfig feature is disabled, skip reconfig processing.");
1926.             return false;
1927.         }
1928.
1929.         InetAddress oldClientAddr = getClientAddress();
1930.
1931.         // update last committed quorum verifier, write the new config to disk
1932.         // and restart leader election if config changed.
1933.         QuorumVerifier prevQV = setQuorumVerifier(qv, true);
1934.
1935.         // There is no log record for the initial config, thus after syncing
1936.         // with leader
1937.         // /zookeeper/config is empty! it is also possible that last committed
1938.         // config is propagated during leader election
1939.         // without the propagation the corresponding log records.
1940.         // so we should explicitly do this (this is not necessary when we're
1941.         // already a Follower/Observer, only
1942.         // for Learner):
1943.         initConfigInZKDatabase();
1944.
1945.         if (prevQV.getVersion() < qv.getVersion() && !prevQV.equals(qv)) {
1946.             Map<Long, QuorumServer> newMembers = qv.getAllMembers();
1947.             updateRemotePeerMXBeans(newMembers);
1948.             if (restartLE) restartLeaderElection(prevQV, qv);
1949.
1950.             QuorumServer myNewQS = newMembers.get(getId());
1951.             if (myNewQS != null && myNewQS.clientAddr != null
1952.                 && !myNewQS.clientAddr.equals(oldClientAddr)) {
1953.                 cnxnFactory.reconfigure(myNewQS.clientAddr);
1954.                 updateThreadName();
1955.             }
1956.
1957.             boolean roleChange = updateLearnerType(qv);
1958.             boolean leaderChange = false;
1959.             if (suggestedLeaderId != null) {
1960.                 // zxid should be non-null too
1961.                 leaderChange = updateVote(suggestedLeaderId, zxid);
1962.             } else {
1963.                 long currentLeaderId = getCurrentVote().getId();
1964.                 QuorumServer myleaderInCurQV = prevQV.getVotingMembers().get(currentLeaderId);
1965.                 QuorumServer myleaderInNewQV = qv.getVotingMembers().get(currentLeaderId);
1966.                 leaderChange = (myleaderInCurQV == null || myleaderInCurQV.addr == null ||
1967.                     myleaderInNewQV == null || !myleaderInCurQV.addr.equals(myleaderInNewQV.addr));
1968.                 // we don't have a designated leader - need to go into leader
1969.                 // election
1970.                 reconfigFlagClear();
1971.             }
1972.
1973.             if (roleChange || leaderChange) {
1974.                 return true;
1975.             }
1976.         }
1977.         return false;
1978.     }
1979. }

```



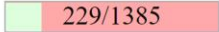
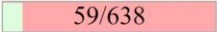
```

1951.     public void delete(final String path, int version, VoidCallback cb,
1952.         Object ctx)
1953.     {
1954.         final String clientPath = path;
1955.         PathUtils.validatePath(clientPath);
1956.
1957.         final String serverPath;
1958.
1959.         // maintain semantics even in chroot case
1960.         // specifically - root cannot be deleted
1961.         // I think this makes sense even in chroot case.
1962.         if (clientPath.equals("/")) {
1963.             // a bit of a hack, but delete() will never succeed and ensures
1964.             // that the same semantics are maintained
1965.             serverPath = clientPath;
1966.         } else {
1967.             serverPath = prependChroot(clientPath);
1968.         }
1969.
1970.         RequestHeader h = new RequestHeader();
1971.         h.setType(ZooDefs.OpCode.delete);
1972.         DeleteRequest request = new DeleteRequest();
1973.         request.setPath(serverPath);
1974.         request.setVersion(version);
1975.         cnxn.queuePacket(h, new ReplyHeader(), request, null, cb, clientPath,
1976.             serverPath, ctx, null);
1977.     }
1978.
1515.     public String create(final String path, byte data[], List<ACL> acl,
1516.         CreateMode createMode)
1517.         throws KeeperException, InterruptedException
1518.     {
1519.         final String clientPath = path;
1520.         PathUtils.validatePath(clientPath, createMode.isSequential());
1521.         EphemeralType.validateTTL(createMode, -1);
1522.
1523.         final String serverPath = prependChroot(clientPath);
1524.
1525.         RequestHeader h = new RequestHeader();
1526.         h.setType(createMode.isContainer() ? ZooDefs.OpCode.createContainer : ZooDefs.OpCode.create);
1527.         CreateRequest request = new CreateRequest();
1528.         CreateResponse response = new CreateResponse();
1529.         request.setData(data);
1530.         request.setFlags(createMode.toFlag());
1531.         request.setPath(serverPath);
1532.         if (acl != null && acl.size() == 0) {
1533.             throw new KeeperException.InvalidACLException();
1534.         }
1535.         request.setAcl(acl);
1536.         ReplyHeader r = cnxn.submitRequest(h, request, response, null);
1537.         if (r.getErr() != 0) {
1538.             throw KeeperException.create(KeeperException.Code.get(r.getErr()),
1539.                 clientPath);
1540.         }
1541.         if (cnxn.chrootPath == null) {
1542.             return response.getPath();
1543.         } else {
1544.             return response.getPath().substring(cnxn.chrootPath.length());
1545.         }
1546.     }
1547.

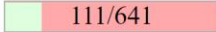
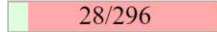
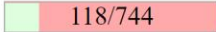
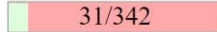
```


Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
2	17%  229/1385	9%  59/638

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
org.apache.zookeeper	1	17%  111/641	9%  28/296
org.apache.zookeeper.server.quorum	1	16%  118/744	9%  31/342

```
1923     public boolean processReconfig(QuorumVerifier qv, Long suggestedLeaderId, Long zxid, boolean restartLE) {
1924 1       if (!isReconfigEnabled()) {
1925           LOG.debug("Reconfig feature is disabled, skip reconfig processing.");
1926 1       return false;
1927     }
1928
1929     InetSocketAddress oldClientAddr = getClientAddress();
1930
1931     // update last committed quorum verifier, write the new config to disk
1932     // and restart leader election if config changed.
1933     QuorumVerifier prevQV = setQuorumVerifier(qv, true);
1934
1935     // There is no log record for the initial config, thus after syncing
1936     // with leader
1937     // /zookeeper/config is empty! it is also possible that last committed
1938     // config is propagated during leader election
1939     // without the propagation the corresponding log records.
1940     // so we should explicitly do this (this is not necessary when we're
1941     // already a Follower/Observer, only
1942     // for Learner):
1943 1     initConfigInZKDatabase();
1944
1945 3     if (prevQV.getVersion() < qv.getVersion() && !prevQV.equals(qv)) {
1946         Map<Long, QuorumServer> newMembers = qv.getAllMembers();
1947 1     updateRemotePeerMXBeans(newMembers);
1948 2     if (restartLE) restartLeaderElection(prevQV, qv);
1949
1950     QuorumServer myNewQS = newMembers.get(getId());
1951 2     if (myNewQS != null && myNewQS.clientAddr != null
1952 1         && !myNewQS.clientAddr.equals(oldClientAddr)) {
1953 1         cnxnFactory.reconfigure(myNewQS.clientAddr);
1954 1     updateThreadName();
1955     }
1956
1957     boolean roleChange = updateLearnerType(qv);
1958     boolean leaderChange = false;
1959 1     if (suggestedLeaderId != null) {
1960         // zxid should be non-null too
1961         leaderChange = updateVote(suggestedLeaderId, zxid);
1962     } else {
1963         long currentLeaderId = getCurrentVote().getId();
1964         QuorumServer myleaderInCurQV = prevQV.getVotingMembers().get(currentLeaderId);
1965         QuorumServer myleaderInNewQV = qv.getVotingMembers().get(currentLeaderId);
1966 3         leaderChange = (myleaderInCurQV == null || myleaderInCurQV.addr == null ||
1967 1             myleaderInNewQV == null || !myleaderInCurQV.addr.equals(myleaderInNewQV.addr));
1968         // we don't have a designated leader - need to go into leader
1969         // election
1970 1     reconfigFlagClear();
1971     }
1972
1973 2     if (roleChange || leaderChange) {
1974 1     return true;
1975     }
1976     }
1977 1     return false;
1978
1979 }
```

```

1951     public void delete(final String path, int version, VoidCallback cb,
1952                        Object ctx)
1953     {
1954         final String clientPath = path;
1955         PathUtils.validatePath(clientPath);
1956
1957         final String serverPath;
1958
1959         // maintain semantics even in chroot case
1960         // specifically - root cannot be deleted
1961         // I think this makes sense even in chroot case.
1962         if (clientPath.equals("/")) {
1963             // a bit of a hack, but delete() will never succeed and ensures
1964             // that the same semantics are maintained
1965             serverPath = clientPath;
1966         } else {
1967             serverPath = prependChroot(clientPath);
1968         }
1969
1970         RequestHeader h = new RequestHeader();
1971         h.setType(ZooDefs.OpCode.delete);
1972         DeleteRequest request = new DeleteRequest();
1973         request.setPath(serverPath);
1974         request.setVersion(version);
1975         cnxn.queuePacket(h, new ReplyHeader(), request, null, cb, clientPath,
1976                        serverPath, ctx, null);
1977     }

```

In tale metodo la coverage è errata poiché è stato eseguito un test case in meno ruga 1962 è coperta

```

1518     {
1519         final String clientPath = path;
1520         PathUtils.validatePath(clientPath, createMode.isSequential());
1521         EphemeralType.validateTTL(createMode, -1);
1522
1523         final String serverPath = prependChroot(clientPath);
1524
1525         RequestHeader h = new RequestHeader();
1526         h.setType(createMode.isContainer() ? ZooDefs.OpCode.createContainer : ZooDefs.OpCode.create);
1527         CreateRequest request = new CreateRequest();
1528         CreateResponse response = new CreateResponse();
1529         request.setData(data);
1530         request.setFlags(createMode.toFlag());
1531         request.setPath(serverPath);
1532         if (acl != null && acl.size() == 0) {
1533             throw new KeeperException.InvalidACLException();
1534         }
1535         request.setAcl(acl);
1536         ReplyHeader r = cnxn.submitRequest(h, request, response, null);
1537         if (r.getErr() != 0) {
1538             throw KeeperException.create(KeeperException.Code.get(r.getErr()),
1539                                       clientPath);
1540         }
1541         if (cnxn.chrootPath == null) {
1542             return response.getPath();
1543         } else {
1544             return response.getPath().substring(cnxn.chrootPath.length());
1545         }
1546     }

```