REPORT ISW2 – Modulo SWTesting

Michela Camilli
0286047

Introduzione

L'obiettivo del progetto è stato quello di applicare le tecniche e gli strumenti di testing, visti durante il corso, a dei progetti software Apache open source. In particolare, sono stati analizzati tre diversi progetti: **JCS (v. 1.3)**, **BOOKKEEPER e ZOOKEEPER**.
Per quanto riguarda JCS, è stato adottato un approccio al testing di tipo white-box; l'analisi relativa a questo progetto è stata, infatti, prettamente improntata ad avere un primo contatto con Junit e con i test parametrici. È stata proposta una nuova implementazione dei test di due classi selezionate, per poi effettuarne un'analisi della copertura strutturale.
Per BookKeeper e ZooKeeper, invece, l'approccio adottato è stato di tipo black-box, andando, per quanto possibile, a reperire le informazioni necessarie alla stesura dei test, tramite la relativa documentazione.
Tutti i progetti sono stati inseriti in una specifica repository on-line su Github, sfruttando le integrazioni con TravisCI e SonarCloud. Inoltre, per l'implementazione dei casi di test (di unità e di integrazione) sono stati utilizzati i framework JUnit 4 e Mockito. Per l'analisi di Statement e Branch Coverage, scelte come metriche di adeguatezza per i test, è stata utlizzata la libreria JaCoCo, mentre, per lo studio della Mutation Coverage, è stata utilizzata la libreria PIT.
A completamento del quadro degli strumenti usati, è stato sfruttato Maven per automatizzare il processo di build e garantire, insieme ai sistemi sovra citati, Continuous Integration e Continuous Testing.

Nel seguito è presentata una breve trattazione delle scelte progettuali adottale per JCS (v.1.3)

JCS (v.1.3)

JCS è un progetto Apache, che consiste di un sistema di caching scritto in Java. È stato sviluppato allo scopo di velocizzare le applicazioni, fornendo un modo per gestire molteplici tipologie di dati in cache. Come scritto precedentemente, però, l'analisi di questo progetto è stata, differentemente dai casi presi in considerazione successivamente, di tipo white-box, adottando un approccio al testing che tenesse in conto solo gli aspetti strutturali e, quindi, relativi al codice.
Di tale progetto è stata considerata una versione precedente, la 1.3; adottare tale distribuzione ha comportato la necessità di gestire, in maniera diretta, lo studio della coverage, richiesto dalle specifiche.
In particolare, tale analisi si basa sull'utilizzo della libreria JaCoCo, che effettua la misurazione della coverage andando ad eseguire delle fasi ben distinte. A causa dell'incompatibilità della versione 1.3 con Maven 3, si è optato per la non inclusione nel repository dei sorgenti del progetto considerato. Per tale ragione, le tre macro-fasi per lo studio della coverage sono state eseguite in locale, slegandole dal sistema di build. All'interno del repository è stato inserito lo script *report.sh* il quale consente, una volta specificati i path necessari, dapprima la creazione, con la CLI di JaCoCo, del fat-jar instrumentato, e poi l'estrazione, sempre da CLI, del report di coverage, che è stato ricavato sia in formato .xml che .csv.
Per quanto concerne, invece, l'obiettivo primario dell'esercizio, le classi di test considerate, in base all'algoritmo, sono state

- *JCSLightLoadUnitTest.java*
- *JCSRemovalSimpleConcurrentTest.java*

In entrambi i casi l'approccio seguito è stato quello di trasformare i test di unità a disposizione in test parametrici. Per fare ciò, all'interno di un metodo di set up è stato istanziato l'oggetto JCS; i parametri da passare sono, quindi, stati legati con un metodo "configure" al runner Parametrized.
È stata, poi, calcolata la coverage di tali test. In particolare, si è ottenuto, come riportato nella Figura 1, un report delle classi coperte. Con i parametri passati, se si considera, ad esempio, la classe *LRUMemoryCache*, come evidenziato dalla Figura 2, si ha un valore di Statement Coverage del 31% e una Branch Coverage del 21%. Aggiungendo ulteriori parametri ai test considerati, però, nella Figura 3, si può già vedere un aumento della Branch Coverage al 22%.

Scelte di Testing

Per quanto concerne la scelta dei test descritti in seguito, questi sono stati identificati secondo un criterio di tipo unidimensionale. In particolare:

- Ogni parametro di input al test è stato considerato in modo indipendente;
- I test sono stati scelti per coprire tutte le classi di equivalenza considerate.

Inotre, per ogni classe considerata, la "reliability" è stata studiata, assumendo profili operazionali con probabilità di utilizzo uniforme, rispetto all'insieme finale dei test.

In seguito, è presentata l'analisi dei progetti Apache BookKeeper e ZooKeeper.

BookKeeper

Apache BookKeeper è un sistema di storage entreprise, ottimizzato per carichi real-time e sviluppato allo scopo di fornire garanzie di scalabilità, consistenza e bassa latenza. La struttura di BookKeeper consta di uno storage di log stream: i dati sono scritti in log come una sequenza di **record** indivisibili; questi ultimi rappresentano l'unità I/O più piccola in BookKeeper.
BookKeeper fornisce due primitive di storage per la rappresentazione dei log: **ledger** e **stream**. Un ledger è una sequenza di entry che può essere terminata sia quando un client la chiude esplicitamente, sia quando un writer subisce un crash. Uno stream, invece, è una sequenza di entry infinita, che, di default, non può essere terminata. BookKeeper replica e memorizza le entry di dati tra diversi server di storage detti **bookie**. Ciascun bookie memorizza porzioni di ledger, invece di immagazzinarli per intero, per garantire migliori performance. Per ciascun ledger L, un *ensamble* è l'insieme dei bookie che memorizzano le entry in L. BookKeeper, inoltre, necessita di un servizio di storage di **metadati** per memorizzare le informazioni relative ai ledger e i bookie disponibili.
Le classi selezionate per le attività di testing sono state: *LedgerMetadataIndex e WriteCache*.

LedgerMetadataIndex

Questa classe, come si evince dalla documentazione ufficiale e dai commenti nel codice, si occupa di mantenere un indice dei metadati dei ledger. In particolare, ci sono due tipologie di metadati che devono essere gestiti in BookKeeper: la *lista dei bookie disponibili* e, per l'appunto, i *ledger metadata*. Questi ultimi possono essere memorizzati in diverse tipologie di storage chiave/valore e sono amministrati da un LedgerManager. La scelta di tale classe è stata dettata proprio da tale interazione con i sistemi di storage, ritenendo che potesse rappresentare un valido esempio di applicazione di test di integrazione. A tale scopo è stata applicata una strategia di tipo top-down, andando ad eseguire test sulle funzionalità più vicine ai confini del SUT ed utilizzando degli stub per simulare i comportamenti mancanti. Tramite l'utilizzo del framework *mockito*, sono stati creati dei mock, i quali hanno consentito la caratterizzazione dell'ambiente di test, indipendentemente da comportamenti esterni alla classe considerata.
Nello specifico, nella classe *LedgerMetadataInedexImpl*, è stato fatto in modo che, ogni qual volta venisse invocato il costruttore della classe under test, si creasse una lista di entry custom su cui poter eseguire test specifici. Per andare, effettivamente, ad eseguire i casi di test è stata inserita, inoltre, una classe *MemoryAppender* con lo scopo di eseguire un check delle invocazioni al log.

Il primo metodo testato è stato:

public LedgerData get(long ledgerId)

Tale metodo preleva un ledger, tramite il suo id, dall'insieme dei ledger memorizzati in un bookie e lo restituisce come risultato. Le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0}. Dato che non è stata trovata documentazione relativa all'impossibilità di inserire valori di ledgerId negativi, sono stati scelti per tale parametro, secondo boundary-value analysis, i valori: {-1, 0, 1}. Una possibile test suite minimale è stata, quindi, considerata, affiancando alla scelta di tali parametri, una possibile selezione della lista di ledger passata in input nel set up di ciascun test. In particolare, i test case scelti sono stati:

- { List<Entry(key = **1**, value = **LedgerData1**)>, ledgerId = **0**}, con valore di ritorno atteso **null**, perché si effettua l'operazione di get su una lista in cui non vi è un ledger con l'id passato;
- { List<Entry(key = -**1**, value =**LedgerData2**)>, ledgerId = **-1**}, con valore di ritorno atteso **LedgerData2**, perché si effettua l'operazione di get su una lista in cui vi è un ledger con l'id passato;
- { List<Entry(key = -**1**, value = **LedgerData2**)>, ledgerId = **-1,** Entry(key = **1**, value = **LedgerData1**)>, ledgerId = **1**}, con valore di ritorno atteso **LedgerData1** perché si effettua l'operazione di get su una lista in cui vi è un ledger con l'id passato.

La test suite ha evidenziato un valore di Statement Coverage del 100%, di Branch Coverage del 75% e un valore di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{2}{2}$ = 100%, come si evince dalle Figure 6 e 7.

Il secondo metodo testato è stato:

public void set(long ledgerId, LedgerData ledgerData)

Tale metodo inserisce un ledger, con uno specifico id, all'interno dell'insieme dei ledger memorizzati in un bookie ed aggiorna le variabili necessarie. Le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0}. Come nel caso precedente, sono stati scelti per tale parametro, secondo boundary-value analysis, i valori: {-1, 0, 1}. Le classi di equivalenza per il valore di LedgerData sono: {null, Valid, Invalid}; tra queste, però, l'unica classe ritenuta effettivamente testabile è stata per istanze Valid di un ledger. Una possibile test suite minimale è stata, quindi, considerata, affiancando alla scelta di tali parametri, una possibile selezione della lista di ledger, passata in input nel set up di ciascun test. In particolare, i test case scelti sono stati:

- { **null**, ledgerId = **1**, ledgerData = **LedgerData1**}, con valore di ritorno atteso **true**, perché si effettua l'operazione di set su una lista vuota in cui non vi è un ledger con l'id passato;
- { List<Entry(key = **0,** value = **LedgerData1**)>, ledgerId = **0**, ledgerData = **LedgerData1**}, con valore di ritorno atteso **false**, perché si effettua l'operazione di set su una lista in cui vi è già un ledger con l'id passato;
- { List<Entry(key = **1**, value = **LedgerData1**)>, ledgerId = **-1,** ledgerData = **LedgerData2**}, con valore di ritorno atteso **true** perché si effettua l'operazione di set su una lista in cui non vi è un ledger con l'id passato.

In cui i valori di ritorno fanno riferimento al check sulla presenza o meno di uno specifico messaggio nel log. La test suite ha evidenziato un valore di Statement Coverage del 100%, di Branch Coverage del 75% e un valore di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{2}{2}$ = 100%, come si evince dalle figure 8 e 9.

Il terzo metodo testato è stato:

public void delete(long ledgerId)

Questo metodo elimina un ledger, tramite il suo id, dall'insieme dei ledger memorizzati nello storage chiave/valore ed aggiorna le variabili necessarie. Le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0}. Come nei casi precedenti, una possibile scelta per tale parametro, secondo boundary-value analysis, è data dai valori: {-1, 0, 1}. Una possibile test suite minimale è stata, quindi, considerata, affiancando alla scelta di tali parametri, una possibile selezione della lista di ledger passata in input nel set up di ciascun test. In particolare, i test case scelti sono stati:

- { **null**, ledgerId = **1**}, con valore di ritorno atteso **false**, perché si effettua l'operazione di delete su una lista vuota;
- { List<Entry(key = **0,** value = **LedgerData1**)>, ledgerId = **0**}, con valore di ritorno atteso **true**, perché si effettua l'operazione di delete su una lista in cui vi è un ledger con l'id passato;
- { List<Entry(key = **1**, value = **LedgerData1**)>, ledgerId = **-1** }, con valore di ritorno atteso **false**, perché si effettua l'operazione di delete su una lista in cui non vi è un ledger con l'id passato.

In cui i valori di ritorno fanno riferimento al check sulla presenza o meno di uno specifico messaggio nel log. La test suite ha evidenziato un valore di Statement Coverage del 100%, di Branch Coverage del 75% e un valore di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{1}{3} \cong$ 33%, come si evince dalle Figure 10 e 11. Tuttavia, entrambe le mutazioni presenti a riga 135 risultano equivalenti rispetto a Strong Mutation in quanto, a parità di input, non cambia l'output del SUT; mentre non sono equivalenti rispetto a Weak Mutation perché gli stati intermedi vengono modificati.

Il successivo metodo testato è stato:

public void setFenced(long ledgerId)

Questo metodo imposta il flag *Fenced* di un ledger. Una volta che il bookie setta "fenced" uno dei suoi ledger, quel bookie non acetta più scritture su quel ledger.
Come nei casi precedenti, le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0}. In questo caso, la test suite minimale considerata è stata:

- { List<Entry(key = **1L,** value = **LedgerData1**(fenced = **true**))>, ledgerId = **0**, ledgerData = **LedgerData1**}, con valore di ritorno atteso **null**, perché si effettua l'operazione di setFenced su una lista in cui vi è già un ledger con flag *Fenced* posto a true;
- { List<Entry(key = -**1L**, value = **LedgerData2**(fenced = **false**))>, ledgerId = **-1L,** ledgerData = **LedgerData2**}, con atteso successo nella modifica del flag, poiché nella lista vi è un ledger, con l'id passato, avente il flag *Fenced* posto a false.

In cui i valori di ritorno fanno riferimento al check sulla presenza o meno di uno specifico messaggio nel log.

Con tali test, il valore di Statement Coverage ottenuto è del 78%, di Branch Coverage del 50% e un valore di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{3}{4}$ = 75%, come riportato nelle Figure 12 e 13. Si è cercato, allo scopo di aumentare tali risultati ottenuti, di aggiungere dei casi di test. In particolare, è stato inserito un test che andasse a settare il flag a true in un ledger in cui questo era posto a false, ma che, prima dell'effettivo setting, venisse eliminato. La build di questo test, però, per motivi di concorrenza, è fallita e, perciò, non è stato possibile calcolare i nuovi valori ottenuti. Il test è stato commentato all'interno del codice.

Il successivo metodo considerato è stato:

public void setExplicitLac(long ledgerId, ByteBuf lac)

Tale metodo imposta l'explicit LAC per un certo ledger. Tale valore è stato introdotto in BookKeeper, come si evince dalla documentazione, per far avanzare periodicamente il valore del LAC, qualora non ci fossero entry aggiunte in modo successivo. Per quanto riguarda la classe di equivalenza del parametro *ledgerId*, sono state fatte le medesime considerazioni dei casi precedenti. Per il parametro *lac*, invece, le classi di equivalenza da considerare, secondo boundary-value analysis, sono: {null, empty, Valid, Invalid}. Tra questi, però, gli unici valori ritenuti testabili sono stati per istanze Valid ed empty di LAC.
In particolare, quindi, la test suite minimale considerata è stata:

- { List<Entry(key = **0,** value = **LedgerData1**(explicitLac = **VALID(0**))>, ledgerId = **0**, explicitLac = **empty**}, con atteso successo nella modifica del LAC, perché vuol dire che una nuova entry è stata aggiunta;
- { List<Entry(key = **-1,** value = **LedgerData1**(explicitLac = **empty**))>, ledgerId = -**1**, explicitLac = **VALID**}, con atteso successo nel setting del LAC.

Il valore di Statement Coverage ottenuto è stato del 97%, mentre quello di Branch Coverage del 50%. Si è raggiunto, poi, un Mutation Score di: $\frac{|D|}{|M|-|E|} = \frac{0}{2}$ = 0%, come risulta dalle Figure 14 e 16. Per aumentare tali valori è stato aggiunto un ulteriore caso di test:

{ List<Entry(key = **1,** value = **LedgerData1**(explicitLac = **empty**))>, ledgerId = -**1**, explicitLac = **VALID**}, in cui il setting del LAC non può avvenire perché si sta cercando di cambiarne il valore per un ledger che non è in lista.

In cui i valori di ritorno fanno riferimento al check sulla presenza o meno di uno specifico messaggio nel log. I nuovi valori ottenuti sono: Statement Coverage del 98%, Branch Coverage del 66% e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{1}{2}$ = 50%, riportato nella Figura 15. Si è cercato di aumentare ulteriormente i valori ottenuti andando a inserire un test che andasse a impostare il LAC in un ledger, ma che venisse eliminato prima dell'effetivo setting. Come nel test precedente, però, per motivi di concorrenza la build è fallita.

Il successivo metodo preso in considerazione è stato:

public void setMasterKey(long ledgerId, byte[] masterKey)

Questo metodo imposta la *MasterKey* per uno specifico ledger. Tale chiave è unica per un certo ledger e, una volta settata, non può essere modificata. Per quanto concerne la classe di equivalenza del parametro *ledgerId*, sono state fatte le medesime considerazioni dei casi precedenti. Per il parametro *masterKey*, invece, le classi di equivalenza da considerare, secondo boundary-value analysis, sono: {null, empty, Valid, Invalid}. Tra questi, però, gli unici valori ritenuti testabili sono stati per istanze Valid ed empty di masterKey.
In particolare, quindi, la test suite minimale considerata è stata:

- { List<Entry(key = **0,** value = **LedgerData1**(masterKey = **empty**))>, ledgerId = **1**, masterKey = **empty**}, con atteso successo nella creazione del ledger e setting di masterKey ad un array vuoto;
- { List<Entry(key = **-1,** value = **LedgerData1**(masterKey = **empty**))>, ledgerId = -**1**, masterKey = **VALID1**}, con atteso successo nel setting di masterKey ad un array valido.

In cui i valori di ritorno fanno riferimento al check sulla presenza o meno di uno specifico messaggio nel log. Il valore di Statement Coverage ottenuto, con tali test, è del 90%, di Branch Coverage del 50% e un valore di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{2}{5}$ = 40%, come evidenziato nelle Figure 17 e 18. Tuttavia, la mutazione presente a riga 196 risulta equivalente rispetto a Strong Mutation in quanto, a parità di input, non cambia l'output del SUT.

L'ultimo test effettuato per questa classe è stato:

public Iterable<Long> getActiveLedgersInRange(final long firstLedgerId, final long lastLedgerId)

Questo metodo preleva i ledger contenuti in un certo range. A tale scopo, con le medesime considerazioni relative ai valori dei due *ledgerId* fatte negli altri test, la test suite scelta è stata:

- { **null**, first = **-1**, last = **1**}, con atteso fallimento della *get* dei ledger, perché la lista è vuota;
- { List<Entry(key = **0,** value = **LedgerData1**)>, first = **-1**, last = **1**}, con atteso successo della *get* e ritorno del valore **LedgerData1**, perché con chiave compressa nell'intervallo;

- { List<Entry(key = **Long.MAX_VALUE,** value = **LedgerData3**)>, first = **-1**, last = **1**}, con atteso fallimento della *get* dei ledger, perché la chiave del ledger presente in lista è maggiore dell'intervallo considerato;
- { List<Entry(key = **1,** value = **LedgerData2**)>, first = **1**, last = -**1**}, con atteso fallimento della *get* dei ledger, perché firstLedgerId maggiore del lastLedgerId.

Il test:

- { List<Entry(key = **1,** value = **LedgerData2**)>, first = **1**, last = **1**}, con atteso successo della *get* dei ledger, perché la chiave del ledger presente in lista è pari all'intervallo considerato.

È fallito, nonostante ritenuto coerente con il fine del metodo testato. Per tale motivo il test è stato commentato nel codice.

Il valore di Statement Coverage ottenuto, con tali test, è del 100% e di Mutation Score dato da: $\frac{|D|}{|M|-|E|} = \frac{1}{1} = 100\%$, come riportato nelle Figure 19 e 20.

WriteCache

Le pagine degli indici dei ledger vengono memorizzate nella cache in un pool di memoria, che consente una gestione più efficiente dello scheduling. Dai commenti del codice considerato, inoltre, si evince che la Write Cache alloca la dimensione richiesta dalla memoria diretta, per poi scomporla in molteplici segmenti. A differenza della classe precedente, in questo caso sono stati eseguiti solamente test d'unità.

Il primo metodo testato è stato:

public ByteBuf get(long ledgerId, long entryId)

Questo metodo preleva una certa entry da un buffer comune, andando a reperire la locazione da un hashmap di indicizzazione. Le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0} e, allo stesso modo, le classi di equivalenza per *entryId* sono: {< 0, >= 0}. Una possibile selezione dei parametri, secondo boundary-value analysis è, sia per *ledgerId* che per *entryId* pari a: {-1, 0, 1}.
Una possibile test suite minimale è stata, quindi, considerata, affiancando alla scelta di tali parametri, una possibile selezione della entry da inserire nel buffer comune nella fase di set up di ciascun test. In particolare:

- { ledgerId = **-1**, entryId = **0**, entry = **null**}, con valore atteso pari a **null**;
- { ledgerId = **0**, entryId = **-1**, entry = **VALID**}, con valore atteso pari a **null**;
- { ledgerId = **1**, entryId = **1**, entry = **VALID**}, con valore atteso pari a alla entry **VALID**.

Con tali parametri, i valori ottenuti sono stati: 100% di Statement Coverage, 100% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{3}{4} = 75\%$, come si evince dalle Figure 23 e 24. Per andare a migliorare quest'ultimo risultato, ed uccidere l'ultima mutazione, è stato aggiunto il test *getDoublePut*, in modo da andare a considerare più segmenti del buffer.
In questo modo si è ottenuto un valore di Mutation Score pari al 100%, come si può vedere dalla Figura 25.

Il secondo metodo testato è stato:

public boolean put(long ledgerId, long entryId, ByteBuf entry)

Questo metodo inserisce una entry in un buffer comune, aggiornando, inoltre, l'hashmap di indicizzazione. Le classi di equivalenza per il valore di *ledgerId* sono: {< 0, >= 0} e, allo stesso modo, le classi di equivalenza per *entryId* sono: {< 0, >= 0}. Per quanto riguarda il parametro *entry*, invece, le possibili classi di equivalenza sono date da: {null, Valid, Invalid}. Gli unici valori ritenuti testabili nell'ultimo caso sono stati: {null, Valid}.
Una possibile test suite minimale è stata, quindi:

- { ledgerId = **-1**, entryId = **0**, entry = **null**}, con atteso **insuccesso** dell'operazione di put;
- { ledgerId = **0**, entryId = **1**, entry = **VALID1**}, con atteso **successo** dell'operazione di put;
- { ledgerId = **1**, entryId = **-1**, entry = **VALID2**}, con atteso **insuccesso** dell'operazione di put.

Sono stati, quindi, calcolati i valori di Coverage, ottenendo: 97% di Statement Coverage e 75% di Branch Coverage. Per quanto riguarda, invece, il Mutation Score, si è ottenuto un risultato pari a $\frac{|D|}{|M|-|E|} = \frac{5}{14} \cong 35\%$. Tali risultati sono riportati nelle Figure 26 e 27.
Aggiungendo, quindi, l' ulteriore caso di test:

- { ledgerId = **1**, entryId = **2**, entry = **VALID3**}, con atteso **successo** dell'operazione di put.

E andando ad inserire nel test la possibilità di aggiungere più volte una entry nella cache, i valori raggiunti sono stati: Statement Coverage pari a 99%, Branch Coverage dell'87% e Mutation Score pari a $\frac{|D|}{|M|-|E|} = \frac{7}{14} = 50\%$, come si evince dalla Figura 28.

ZooKeeper

ZooKeeper è un servizio centralizzato per il mantenimento delle informazioni di configurazione, del naming, garantendo sincronizzazione distribuita e servizi di gruppo. Espone un semplice insieme di primitive su cui le applicazioni distribuite possono basarsi per implementare tali servizi. È progettato per essere facile da programmare; consente ai processi distribuiti di coordinarsi tra loro attraverso uno spazio dei nomi gerarchico condiviso, organizzato in modo simile a un file system standard. Il namespace è costituito da registri di dati - chiamati **znodes**, nel gergo ZooKeeper - e questi sono simili a file e directory. A differenza di un tipico file system, progettato per l'archiviazione, i dati di ZooKeeper vengono mantenuti in memoria, il che significa che ZooKeeper può ottenere throughput elevato e bassa latenza. Come i processi distribuiti che coordina, ZooKeeper stesso è pensato per essere replicato su un insieme di host chiamato **ensemble**. I server che compongono il servizio ZooKeeper devono conoscersi tra loro. Mantengono un'immagine di stato in memoria, insieme a **registri delle transazioni** e **snapshot** in un archivio permanente. Finché la maggior parte dei server sarà disponibile, sarà disponibile il servizio ZooKeeper.
Le classi selezionate per le attività di testing sono state: *ZKUtil* e *FileTxnLog.*

ZKUtil

Tale classe contiene delle utils necessarie a garantire il corretto funzionamento della struttura gerarchica di ZooKeeper. In particolare, contiene metodi atti all'attraversamento, alla cancellazione e alla validazione del sistema, a partire da uno specifico pathRoot, per tutto il subtree di znode. Inoltre, contiene dei metodi necessari alla gestione delle ACL. Questa classe ha rappresentato un valido esempio di applicazione del framework *mockito*. Infatti, si è cercato di testare questa classe indipendentemente dai comportamenti delle classi esterne alla stessa, realizzando, quindi, la base di un test di integrazione con strategia top-down. A tale scopo sono stati creati dei mock, che permettessero la creazione di subtree custom, su cui andare, quindi, ad eseguire test con i valori di ritorno voluti. In particolare, è stata creata una classe *NodeTree*, per la creazione di un subtree, a partire da una specifica root.

Il primo metodo testato è stato:

public static List<String> listSubTreeBFS(ZooKeeper zk, final String pathRoot)

Tale metodo, come si legge dai commenti al codice, esegue un attraversamento del sistema di tipo BFS, partendo da un certo pathRoot. Con BFS si intende l'attraversamento di un tree un livello dopo l'altro, partendo, prima, dal livello dei children nodes, per poi passare ai grandchildren nodes e così via, fino al raggiungimento dei nodi foglia. Si è testato, quindi, che dato un attraversamento di tipo BFS a partire dalle radici passate come parametro, si ottenesse lo stesso risultato con la chiamata del metodo considerato.
È stato, quindi, effettuato il mock della classe ZooKeeper nel set up del test, per poi eseguire un'implementazione custom del metodo *getChildren*.
Le classi di equivalenza per il valore di *pathRoot* sono: {null, empty, Valid, Invalid}. Al fine del testing di questo metodo, solo gli ultrimi tre valori sono stati ritenuti interessanti.
Una possibile test suite minimale è stata, quindi:

- { pathRoot = **VALID }**, avente come input un subtree con due children nodes, con attesa uguaglianza dell'attraversamento BFS di tale subtree;
- { pathRoot = **empty** }, avente come input un subtree con un children node, con attesa uguaglianza dell'attraversamento BFS di tale subtree;
- { pathRoot = **INVALID** }, con atteso sollevamento di un'eccezione.

Da documentazione, i possibili valori **non validi** per il path name sono:

- Il carattere nullo (\u0000), il quale è stato selezionato per questo test;
- I caratteri \u0001 - \u0019 e \u007F - \u009F;
- I caratteri " - \uF8FFF, \uFFF0 - \uFFFF, \uXFFFE - \uXFFFF e \uF0000 - \uFFFFF;
- Il carattere "." può essere usato come parte di un altro nome, ma "." e ".." non possono essere usati da soli;
- Il token "zookeeper" è riservato.

Con tali parametri, i valori ottenuti sono stati: 100% di Statement Coverage, 100% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{3}{3}$ = 100%, come si evince dalle Figure 30 e 31.

Il secondo metodo testato è stato:

public static void visitSubTreeDFS(ZooKeeper zk, final String pathRoot, boolean watch, StringCallback cb)

Tale metodo permette di visitare il subtree, con la radice come path iniziale. Chiama, inoltre, una callback per ciascuno znode trovato durante la ricerca. Esegue un attraversamento di tipo depth-first del subtree.
Il test è consistito nel verificare se, dato un attraversamento DFS del subtree stabilito, il numero di callback effettuate coincidesse con il valore atteso. Anche questo caso, il test è stato reso possibile dall'utilizzo di mock, questa volta sia per la classe ZooKeeper che per StringCallback.

Le classi di equivalenza per il valore di *pathRoot* sono: {null, empty, Valid, Invalid}. Al fine del testing di questo metodo, solo gli ultrimi tre valori sono stati ritenuti interessanti.
Una possibile test suite minimale è stata, quindi:

- { pathRoot = **VALID }**, avente come input un subtree con due children nodes, con numero di callback atteso pari a 3;
- { pathRoot = **empty** }, avente come input un subtree con un children node, con numero di callback atteso pari a 2;
- { pathRoot = **INVALID** }, con atteso sollevamento di un'eccezione.

In questo caso, i valori di stringa non valida selezionati sono stati: "." e "\ud800".
Con questi test, i valori ottenuti sono stati: 97% di Statement Coverage, 100% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{4}{9} \cong 44\%$, come si evince dalle Figure 32 e 33. Per aumentare tale valore, è stato aggiunto un ulteriore caso di test, che va ad aggiungere un subree avente due children nodes e due grandchildren nodes. Questo ha permesso di ottenere un nuovo Mutation Score uguale a: $\frac{|D|}{|M|-|E|} = \frac{7}{9} \cong$ 77%, come mostrato nella figura 34.

Il terzo metodo testato è stato:

public static String validateFileInput(String filePath)

Questo metodo rappresenta una utiliy necessaria alla validazione del file path. Ritorna null se è valido, altrimenti ritorna un messaggio di errore. Sono stati creati tre test, ciascuno adibito, rispettivamente, a testare se il file: esiste, ha i permessi di lettura o è una directory. La test suite minimale per tutti i casi è stata:
Per *exists*:

- { filePath = **VALID }**, con atteso valore **null**, nel caso in cui il file sia stato creato;
- { filePath = **VALID }**, con atteso messaggio di errore, nel caso in cui il file non sia stato creato.

Per *canRead*:

- { filePath = **VALID }**, con atteso valore **null**, nel caso in cui il file sia stato reso readable;
- { filePath = **VALID }**, con atteso messaggio di errore, nel caso in cui il file non sia stato reso readable.

Per *isDirectory*:

- { filePath = **VALID }**, con atteso valore **null**, nel caso in cui il file sia una directory;
- { filePath = **VALID }**, con atteso messaggio di errore, nel caso in cui il file non sia una directory.

Con questi test, i valori ottenuti sono stati:100% di Statement Coverage, 100% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{7}{7}$ = 100%, come mostrato nelle Figure 35 e 36.

Il successivo metodo testato è stato:

public static String aclToString(List<ACL> acls)

Questo metodo prende in input una lista di ACL, per poi trasformarle in una stringa di struttura predefinita.
Per quanto riguarda le ACL builtin in ZookKepper, ve ne sono principalente cinque:

- **world**, ha un singolo id, *anyone*, che rappresenta tutti;
- **auth**, non usa id, rappresenta ogni utente autenticato;
- **digest**, usa una stringa *username:password* per generare un hash che viene usato come ACL ID;
- **host**, usa l'host name del client come ACL ID;
- **ip**, usa l'host IP del client come ACL ID.

Nel caso in esame, si è scelto come ACL il tipo **digest**.
Le classi di equivalenza considerate per la List<ACL> sono state: {null, Valid}. In particolare, la test suite minimale considerata è stata:

- { acls = **List<ACL(digest)>** }, con atteso valore di ritorno dati dalla stringa voluta;
- { acls = **null** }, con atteso valore di ritorno dati dalla stringa vuota.

Con questi test, i valori ottenuti sono stati: 100% di Statement Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{1}{1}$ = 100%, come riportato nelle figure 37 e 38.

L'ultimo metodo considerato è stato:

public static String getPermString(int perms)

Tale metodo crea la stringa relativa all'insieme dei permessi in input. In ZooKeeper ciascun permesso è associato ad un numero intero che, posto in and bitwise con *perms*, dice qual è la specifica stringa associata. Le classi di equivalenza per il valore di *perms* sono: {< 0, >= 0}. Dato che non è stata trovata documentazione relativa all'impossibilità di inserire valori di perms negativi, sono stati scelti per tale parametro, secondo boundary-value analysis, i valori: {-1, 0, 1}.
In particolare, la test suite minimale considerata è stata:

- { perms = **1** }, con valore di ritorno atteso “**r**”;
- { perms = **0** }, con valore di ritorno atteso “”;
- { perms = **-1** }, con valore di ritorno atteso “”. Infatti, settando -1 come parametro, si è ritenuto si dovesse ottenere una stringa vuota come risultato; invece, si ottiene la stringa relativa al permesso “ALL” che, da documentazione, è impostato a 31. Questo è dato dal fatto che, usando l’operatore bitwise di Java, anche -1 è visto come 11111.

Quest’ultimo test è stato quindi commentato prima della build.

Con questi test, i valori ottenuti sono stati: 100% di Statement Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{11}{11}$ = 100%, come è mostrato nelle Figure 39 e 40.

FileTxnLog

Come si evince dalla documentazione, questa classe implementa l’interfaccia TxnLog e fornisce le API per accedere ai log transazionali ed aggiungervi entry. La Data Directory di ZooKeeper contiene file che rappresentano una copia persistente dei znode, memorizzati da un certo ensamble. Questi file si dividono in *Snapshot* e *Log File Transazionali*. Non appena vengono apportate delle modifiche ai znode, queste vengono aggiunte ad un log della transazione.
A differenza della classe precedente, in questo caso sono stati eseguiti solamente test d’unità.

Il primo metodo testato è stato:

public boolean truncate(long zxid)

Questo metodo, da documentazione, serve a troncare i long transazionali correnti. In particolare, *zxid* rappresenta l’id della transazione in ZooKeeper e il valore di ritorno è true se l’operazione avviene con successo e falso altrimenti. Al fine di testare questo metodo, è stato inserito un nuovo record nel log, per poi constatare se, effettivamente, l’operazione di truncate venisse eseguita. Le classi di equivalenza per il valore di *zxid* sono: {< 0, >= 0}. Dato che non è stata trovata documentazione relativa all’impossibilità di inserire valori di zxid negativi, sono stati scelti per tale parametro, secondo boundary-value analysis, i valori: {-1, 0, 1}. In particolare, la test suite minimale considerata è stata:

- { zxid = **1** }, con atteso **successo** della truncate;
- { zxid = **0** }, con atteso **successo** della truncate;
- { zxid = **-1** }, con atteso **successo** della truncate.

Con questi test, i valori ottenuti sono stati: 76% di Statement Coverage, 66% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{1}{4}$ = 25%, come riportato nelle Figure 43 e 45. Per quanto riguarda, però, le mutazioni che non sono state uccise, risultano equivalenti rispetto a Strong Mutation in quanto, a parità di input, non cambia l’output del SUT. Per aumentare, invece, i valori di Coverage, è stato aggiunto un ulteriore test che va ad eliminare la directory precedentemente create per il log, per poi chiamare la truncate sul medesimo log, in modo da causarne il fallimento. Con l’aggiunta di tale test si sono ottenuti i nuovi valori pari a: 87% di Statement Coverage e 83% di Branch Coverage, come mostrato nelle Figura 44.

Il secondo metodo testato è stato:

public TxnIterator read(long zxid, boolean fastForward)

Tale metodo, come si può leggere dai commenti al codice, legge tutte le transazioni a partire da un certo zxid. In particolare, quindi, lo *zxid* rappresenta l’id della transazione da cui iniziare la lettura; il parametro *fastForward*, invece, se posto a true sigifica che l’iterator dovrebbe essere mandato in avanti fino a puntare alla transazione di un certo zxid, altrimenti l’iterator punterà alla transazione di partenza di un txnlog, che potrebbe contenere o meno la transazione di un certo zxid. Al fine di testare questo metodo, è stato aggiunto un certo numero di record nel log, per poi constatare se, dato l’iterator ottenuto dall’invocazione del metodo, effettivamente fossero presenti tanti record quanti attesi. Le classi di equivalenza per il valore di *zxid* sono: {< 0, >= 0}, mentre, per *fastForward* sono: {true, false}. Con le medesime considerazioni fatte nel test precedente, sono stati scelti per il parametro *zxid*, secondo boundary-value analysis, i valori: {-1, 0, 1}. In particolare, la test suite minimale considerata è stata:

- { zxid = **0**, fastForward = **true** }, con numero di record inseriti pari a 3 e con attesa read per la specifica transazione;
- { zxid = **1**, fastForward = **false** }, con numero di record inseriti pari a 1 e con attesa read per la specifica transazione;
- { zxid = **-1**, fastForward = **true** }, con numero di record inseriti pari a 0 e con attesa read di 0 record per la transazione.

Con questi test, i valori ottenuti sono stati: 100% di Statement Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{2}{2}$ = 100%, come mostrato nelle Figure 46 e 47.

L'ultimo metodo testato è stato:

public synchronized boolean append(TxnHeader hdr, Record txn, TxnDigest digest)

Questo metodo permette di aggiungere una entry ad un log transazionale. Ha come parametri: *hdr*, che rappresenta l'header della transazione; *txn*, che rappresenta il record della transazione e *digest*, che rappresenta il digest della transazione. Il metodo ritorna true se la append va a buon fine e false altrimenti. Le classi di equivalenza dell'oggetto hdr sono: {null, Valid, Invalid}, per l'oggetto txn sono: {null, Valid, Invalid} e lo stesso vale per il digest. In tutti i casi non si è ritenuto necessario considerare il valore "Invalid" perché non interessante per il test. La test suite minimale considerata è stata, quindi:

- { hdr = **VALID**, txn = **VALID**, digest = **VALID** }, con atteso successo della append;
- { hdr = **null**, txn = **VALID**, digest = **VALID** }, con atteso fallimento della append;
- { hdr = **VALID**, txn = **null**, digest = **VALID** }, con atteso successo della append.

Con questi test, i valori ottenuti sono stati: 96% di Statement Coverage, 80% di Branch Coverage e Mutation Score pari a: $\frac{|D|}{|M|-|E|} = \frac{12}{16}$ = 75%, come riportato nelle Figure 48 e 49. Per quanto riguarda le mutazioni a riga 272, 291 e 292, inoltre, possono essere considerate equivalenti rispetto a Strong Mutation in quanto, a parità di input, non cambia l'output del SUT.

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.jcs.auxiliary.remote		0%		0%	436	436	1.161	1.161	195	195	22	22
org.apache.jcs.auxiliary.disk.indexed		0%		0%	227	227	779	779	90	90	12	12
org.apache.jcs.auxiliary.remote.server		0%		0%	260	260	576	576	93	93	7	7
org.apache.jcs.auxiliary.disk.block		0%		0%	184	184	596	596	91	91	12	12
org.apache.jcs.engine		12%		1%	256	291	619	725	156	191	24	27
org.apache.jcs.utils.struct		7%		7%	187	204	507	567	63	72	7	9
org.apache.jcs.auxiliary.disk.jdbc		0%		0%	163	163	585	585	90	90	9	9
org.apache.jcs.auxiliary.lateral.socket.tcp		0%		0%	186	186	547	547	96	96	9	9
org.apache.jcs.engine.control		31%		19%	215	272	448	687	50	98	0	3
org.apache.jcs.auxiliary.lateral		0%		0%	169	169	472	472	108	108	13	13
org.apache.jcs.auxiliary.lateral.socket.tcp.discovery		0%		0%	110	110	375	375	61	61	10	10
org.apache.jcs.engine.memory.lru		30%		21%	68	83	157	248	22	32	2	3
org.apache.jcs.auxiliary.disk.jdbc.mysql		0%		0%	71	71	190	190	33	33	6	6
org.apache.jcs.auxiliary.disk		0%		0%	72	72	222	222	46	46	6	6
org.apache.jcs.engine.memory.mru		0%		0%	46	46	179	179	13	13	1	1
org.apache.jcs.config		32%		30%	54	76	137	211	9	23	1	3
org.apache.jcs.utils.threadpool		43%		24%	51	69	127	213	18	36	2	4
org.apache.jcs.admin		0%		0%	35	35	95	95	26	26	4	4
org.apache.jcs.access		17%		17%	44	54	94	117	31	40	0	2
org.apache.jcs.engine.memory.shrinking		0%		0%	35	35	87	87	6	6	1	1
org.apache.jcs.utils.access		0%		0%	24	24	61	61	9	9	2	2
org.apache.jcs.engine.control.event		23%		13%	29	35	74	98	11	17	3	6
org.apache.jcs.auxiliary.disk.jdbc.hsql		0%		0%	15	15	70	70	7	7	1	1
org.apache.jcs.access.monitor		0%		0%	20	20	56	56	6	6	2	2
org.apache.jcs.engine.control.group		0%		0%	22	22	30	30	8	8	2	2
org.apache.jcs.utils.serialization		0%		0%	13	13	51	51	7	7	2	2
org.apache.jcs.auxiliary.disk.jdbc.mysql.util		0%		0%	12	12	37	37	4	4	2	2
org.apache.jcs.utils.servlet		0%		0%	21	21	37	37	9	9	2	2
org.apache.jcs.engine.memory		22%		14%	21	24	39	51	14	17	1	2
org.apache.jcs.utils.props		0%		0%	14	14	34	34	2	2	1	1
org.apache.jcs.admin.servlet		0%		0%	12	12	30	30	2	2	1	1
org.apache.jcs.access.exception		0%		n/a	17	17	34	34	17	17	8	8
org.apache.jcs.utils.config		0%		0%	5	5	15	15	2	2	2	2
org.apache.jcs.auxiliary		0%		0%	13	13	22	22	10	10	1	1
org.apache.jcs.auxiliary.lateral.socket.tcp.utils		0%		n/a	4	4	17	17	4	4	1	1
org.apache.jcs.utils.timing		0%		0%	6	6	15	15	5	5	2	2
org.apache.jcs.utils.net		0%		0%	5	5	10	10	3	3	1	1
org.apache.jcs.engine.stats		89%		75%	10	26	7	53	7	20	0	3
org.apache.jcs		70%		75%	2	8	4	16	1	6	0	1
org.apache.jcs.auxiliary.remote.behavior		0%		0%	2	2	1	1	1	1	1	1
org.apache.jcs.engine.memory.util		100%		n/a	0	1	0	3	0	1	0	1
Total	38.783 of 41.809	7%	3.306 of 3.490	5%	3.136	3.342	8.597	9.373	1.426	1.597	183	207

Figura 1: JaCoCo Coverage

org.apache.jcs.engine.memory.lru

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
LRUMemoryCache		31%		21%	56	71	140	231	10	20	0	1
LRUMemoryCache.IteratorWrapper		0%		n/a	6	6	9	9	6	6	1	1
LRUMemoryCache.MapEntryWrapper		0%		n/a	6	6	8	8	6	6	1	1
Total	895 of 1.282	30%	80 of 102	21%	68	83	157	248	22	32	2	3

Figura 2: JaCoCo Coverage ***LRU*** prima dell'aggiunta di ulteriori test

org.apache.jcs.engine.memory.lru

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
LRUMemoryCache		31%		22%	55	71	140	231	10	20	0	1
LRUMemoryCache.IteratorWrapper		0%		n/a	6	6	9	9	6	6	1	1
LRUMemoryCache.MapEntryWrapper		0%		n/a	6	6	8	8	6	6	1	1
Total	895 of 1.282	30%	79 of 102	22%	67	83	157	248	22	32	2	3

Figura 3: JaCoCo Coverage ***LRU*** dopo l'aggiunta di ulteriori test

LedgerMetadataIndex

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
removeDeletedLedgers()		0%		0%	5	5	17	17	1	1
flush()		0%		0%	3	3	14	14	1	1
setFenced(long)		78%		50%	3	5	3	13	0	1
lambda$delete$1(long, Map.Entry)		0%		0%	2	2	1	1	1	1
lambdanew0()		0%		n/a	1	1	1	1	1	1
close()		0%		n/a	1	1	2	2	1	1
setExplicitLac(long, ByteBuf)		97%		50%	3	4	1	10	0	1
setMasterKey(long, byte[])		100%		71%	4	8	0	19	0	1
LedgerMetadataIndex(ServerConfiguration, KeyValueStorageFactory, String, StatsLogger)		100%		100%	0	2	0	17	0	1
set(long, DbLedgerStorageDataFormats.LedgerData)		100%		75%	1	3	0	8	0	1
delete(long)		100%		75%	1	3	0	7	0	1
get(long)		100%		75%	1	3	0	6	0	1
getActiveLedgersInRange(long, long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	142 of 508	72%	28 of 56	50%	25	42	38	116	5	14

Figura 4: LedgerMetadataIndex Coverage Suite Minimale

LedgerMetadataIndex

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
removeDeletedLedgers()		0%		0%	5	5	17	17	1	1
flush()		0%		0%	3	3	14	14	1	1
setFenced(long)		78%		50%	3	5	3	13	0	1
lambda$delete$1(long, Map.Entry)		0%		0%	2	2	1	1	1	1
lambdanew0()		0%		n/a	1	1	1	1	1	1
close()		0%		n/a	1	1	2	2	1	1
setExplicitLac(long, ByteBuf)		98%		66%	2	4	1	11	0	1
setMasterKey(long, byte[])		100%		71%	4	8	0	19	0	1
LedgerMetadataIndex(ServerConfiguration, KeyValueStorageFactory, String, StatsLogger)		100%		100%	0	2	0	17	0	1
set(long, DbLedgerStorageDataFormats.LedgerData)		100%		75%	1	3	0	8	0	1
delete(long)		100%		75%	1	3	0	7	0	1
get(long)		100%		75%	1	3	0	6	0	1
getActiveLedgersInRange(long, long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	142 of 518	72%	27 of 56	51%	24	42	38	117	5	14

Figura 5: LedgerMetadataIndex Coverage dopo l'aggiunta di ulteriori test

```
100.    public LedgerData get(long ledgerId) throws IOException {
101.        LedgerData ledgerData = ledgers.get(ledgerId);
102.        if (ledgerData == null) {
103.            if (log.isDebugEnabled()) {
104.                log.debug("Ledger not found {}", ledgerId);
105.            }
106.            throw new Bookie.NoLedgerException(ledgerId);
107.        }
108.
109.        return ledgerData;
110.    }
```

Figura 6: LedgerMetadataIndex ***get*** Coverage

```
100      public LedgerData get(long ledgerId) throws IOException {
101          LedgerData ledgerData = ledgers.get(ledgerId);
102 1        if (ledgerData == null) {
103              if (log.isDebugEnabled()) {
104                  log.debug("Ledger not found {}", ledgerId);
105              }
106              throw new Bookie.NoLedgerException(ledgerId);
107          }
108
109 1        return ledgerData;
110      }
```

Figura 7: LedgerMetadataIndex ***get*** Mutation

```
112.    public void set(long ledgerId, LedgerData ledgerData) throws IOException {
113.        ledgerData = LedgerData.newBuilder(ledgerData).setExists(true).build();
114.
115.        if (ledgers.put(ledgerId, ledgerData) == null) {
116.            if (log.isDebugEnabled()) {
117.                log.debug("Added new ledger {}", ledgerId);
118.            }
119.            ledgersCount.incrementAndGet();
120.        }
121.
122.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
123.        pendingDeletedLedgers.remove(ledgerId);
124.    }
```

Figura 8: LedgerMetadataIndex ***set*** Coverage

```
112      public void set(long ledgerId, LedgerData ledgerData) throws IOException {
113          ledgerData = LedgerData.newBuilder(ledgerData).setExists(true).build();
114
115 1        if (ledgers.put(ledgerId, ledgerData) == null) {
116              if (log.isDebugEnabled()) {
117                  log.debug("Added new ledger {}", ledgerId);
118              }
119              ledgersCount.incrementAndGet();
120          }
121
122          pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
123          pendingDeletedLedgers.remove(ledgerId);
124      }
```

Figura 9: LedgerMetadataIndex ***set*** Mutation

```
126.    public void delete(long ledgerId) throws IOException {
127.        if (ledgers.remove(ledgerId) != null) {
128.            if (log.isDebugEnabled()) {
129.                log.debug("Removed ledger {}", ledgerId);
130.            }
131.            ledgersCount.decrementAndGet();
132.        }
133.
134.        pendingDeletedLedgers.add(ledgerId);
135.        pendingLedgersUpdates.removeIf(e -> e.getKey() == ledgerId);
136.    }
```

Figura 10: LedgerMetadataIndex ***delete*** Coverage

```
126      public void delete(long ledgerId) throws IOException {
127 1        if (ledgers.remove(ledgerId) != null) {
128              if (log.isDebugEnabled()) {
129                  log.debug("Removed ledger {}", ledgerId);
130              }
131              ledgersCount.decrementAndGet();
132          }
133
134          pendingDeletedLedgers.add(ledgerId);
135 2        pendingLedgersUpdates.removeIf(e -> e.getKey() == ledgerId);
136      }
```

Figura 11: LedgerMetadataIndex ***delete*** Mutation

```
147.
148.    public boolean setFenced(long ledgerId) throws IOException {
149.        LedgerData ledgerData = get(ledgerId);
150.        if (ledgerData.getFenced()) {
151.            return false;
152.        }
153.
154.        LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
155.
156.        if (ledgers.put(ledgerId, newLedgerData) == null) {
157.            // Ledger had been deleted
158.            if (log.isDebugEnabled()) {
159.                log.debug("Re-inserted fenced ledger {}", ledgerId);
160.            }
161.            ledgersCount.incrementAndGet();
162.        } else {
163.            if (log.isDebugEnabled()) {
164.                log.debug("Set fenced ledger {}", ledgerId);
165.            }
166.        }
167.
168.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
169.        pendingDeletedLedgers.remove(ledgerId);
170.        return true;
171.    }
```

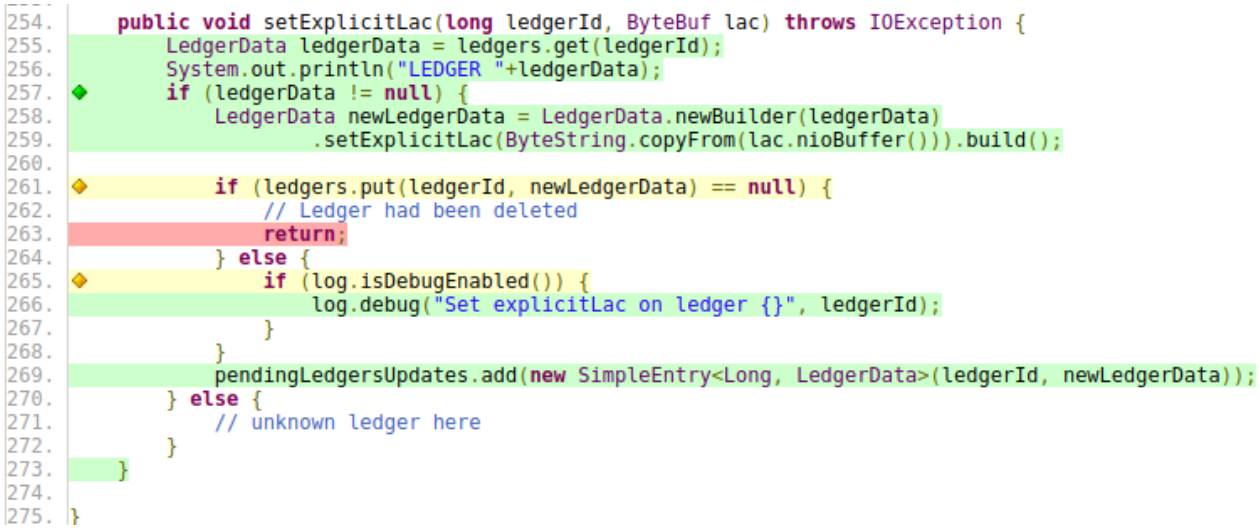
Figura 12: LedgerMetadataIndex ***setFenced*** Coverage

```
148      public boolean setFenced(long ledgerId) throws IOException {
149          LedgerData ledgerData = get(ledgerId);
150 1        if (ledgerData.getFenced()) {
151 1            return false;
152          }
153
154          LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
155
156 1        if (ledgers.put(ledgerId, newLedgerData) == null) {
157              // Ledger had been deleted
158              if (log.isDebugEnabled()) {
159                  log.debug("Re-inserted fenced ledger {}", ledgerId);
160              }
161              ledgersCount.incrementAndGet();
162          } else {
163              if (log.isDebugEnabled()) {
164                  log.debug("Set fenced ledger {}", ledgerId);
165              }
166          }
167
168          pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
169          pendingDeletedLedgers.remove(ledgerId);
170 1        return true;
171      }
```

Figura 13: LedgerMetadataIndex ***setFenced*** Mutation

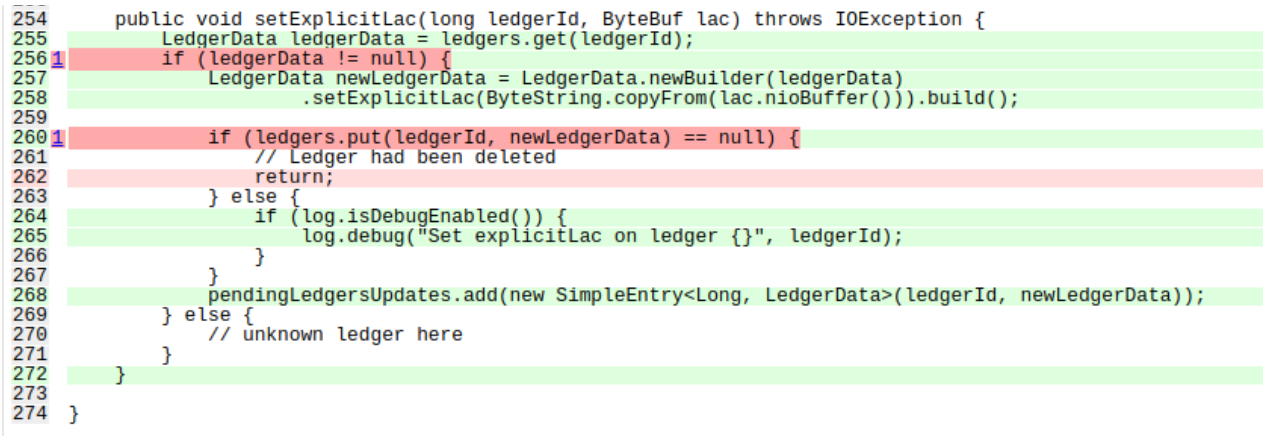
```
    public void setExplicitLac(long ledgerId, ByteBuf lac) throws IOException {
        LedgerData ledgerData = ledgers.get(ledgerId);
        if (ledgerData != null) {
            LedgerData newLedgerData = LedgerData.newBuilder(ledgerData)
                    .setExplicitLac(ByteString.copyFrom(lac.nioBuffer())).build();

            if (ledgers.put(ledgerId, newLedgerData) == null) {
                // Ledger had been deleted
                return;
            } else {
                if (log.isDebugEnabled()) {
                    log.debug("Set explicitLac on ledger {}", ledgerId);
                }
            }
            pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
        } else {
            // unknown ledger here
        }
    }

}
```

Figura 14: LedgerMetadataIndex ***setExplicitLac*** Coverage Suite Minimale



```
254.    public void setExplicitLac(long ledgerId, ByteBuf lac) throws IOException {
255.        LedgerData ledgerData = ledgers.get(ledgerId);
256.        System.out.println("LEDGER "+ledgerData);
257.        if (ledgerData != null) {
258.            LedgerData newLedgerData = LedgerData.newBuilder(ledgerData)
259.                    .setExplicitLac(ByteString.copyFrom(lac.nioBuffer())).build();
260.
261.            if (ledgers.put(ledgerId, newLedgerData) == null) {
262.                // Ledger had been deleted
263.                return;
264.            } else {
265.                if (log.isDebugEnabled()) {
266.                    log.debug("Set explicitLac on ledger {}", ledgerId);
267.                }
268.            }
269.            pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
270.        } else {
271.            // unknown ledger here
272.        }
273.    }
274.
275. }
```

Figura 15: LedgerMetadataIndex ***setExplicitLac*** Coverage dopo l'aggiunta di ulteriori test



```
254     public void setExplicitLac(long ledgerId, ByteBuf lac) throws IOException {
255         LedgerData ledgerData = ledgers.get(ledgerId);
256 1       if (ledgerData != null) {
257             LedgerData newLedgerData = LedgerData.newBuilder(ledgerData)
258                     .setExplicitLac(ByteString.copyFrom(lac.nioBuffer())).build();
259
260 1           if (ledgers.put(ledgerId, newLedgerData) == null) {
261                 // Ledger had been deleted
262                 return;
263             } else {
264                 if (log.isDebugEnabled()) {
265                     log.debug("Set explicitLac on ledger {}", ledgerId);
266                 }
267             }
268             pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
269         } else {
270             // unknown ledger here
271         }
272     }
273
274 }
```

Figura 16: LedgerMetadataIndex ***setExplicitLac*** Mutation

```
173.    public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
174.        LedgerData ledgerData = ledgers.get(ledgerId);
175.        if (ledgerData == null) {
176.            // New ledger inserted
177.            ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
178.                    .setMasterKey(ByteString.copyFrom(masterKey)).build();
179.            if (log.isDebugEnabled()) {
180.                log.debug("Inserting new ledger {}", ledgerId);
181.            }
182.        } else {
183.            byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
184.            if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
185.                // update master key of the ledger
186.                ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
187.                if (log.isDebugEnabled()) {
188.                    log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
189.                }
190.            } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
191.                log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
192.                throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
193.            }
194.        }
195.
196.        if (ledgers.put(ledgerId, ledgerData) == null) {
197.            ledgersCount.incrementAndGet();
198.        }
199.
200.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
201.        pendingDeletedLedgers.remove(ledgerId);
202.    }
```

Figura 17: LedgerMetadataIndex ***setMasterKey*** Coverage

```
173     public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
174         LedgerData ledgerData = ledgers.get(ledgerId);
1751        if (ledgerData == null) {
176             // New ledger inserted
177             ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
178                     .setMasterKey(ByteString.copyFrom(masterKey)).build();
179             if (log.isDebugEnabled()) {
180                 log.debug("Inserting new ledger {}", ledgerId);
181             }
182         } else {
183             byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
1841            if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
185                 // update master key of the ledger
186                 ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
187                 if (log.isDebugEnabled()) {
188                     log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
189                 }
1902             } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
191                 log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
192                 throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
193             }
194         }
195
1961        if (ledgers.put(ledgerId, ledgerData) == null) {
197             ledgersCount.incrementAndGet();
198         }
199
200         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
201         pendingDeletedLedgers.remove(ledgerId);
202     }
```

Figura 18: LedgerMetadataIndex ***setMasterKey*** Mutation

```
138.    public Iterable<Long> getActiveLedgersInRange(final long firstLedgerId, final long lastLedgerId)
139.            throws IOException {
140.        return Iterables.filter(ledgers.keys(), new Predicate<Long>() {
141.            @Override
142.            public boolean apply(Long ledgerId) {
143.                return ledgerId >= firstLedgerId && ledgerId < lastLedgerId;
144.            }
145.        });
146.    }
```

Figura 19: LedgerMetadataIndex ***getActiveLedgersInRange*** Coverage

```
138     public Iterable<Long> getActiveLedgersInRange(final long firstLedgerId, final long lastLedgerId)
139             throws IOException {
1401        return Iterables.filter(ledgers.keys(), new Predicate<Long>() {
141             @Override
142             public boolean apply(Long ledgerId) {
143                 return ledgerId >= firstLedgerId && ledgerId < lastLedgerId;
144             }
145         });
146     }
```

Figura 20: LedgerMetadataIndex ***getActiveLedgersInRange*** Mutation

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda$forEach$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)		0%		n/a	1	1	2	2	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		97%		75%	2	5	2	20	0	1
get(long, long)		100%		100%	0	2	0	10	0	1
clear()		100%		n/a	0	1	0	7	0	1
close()		100%		100%	0	2	0	3	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
count()		100%		n/a	0	1	0	1	0	1
Total	266 of 617	56%	24 of 38	36%	21	35	52	120	7	16

Figura 21: WriteCache Coverage Suite Minimale

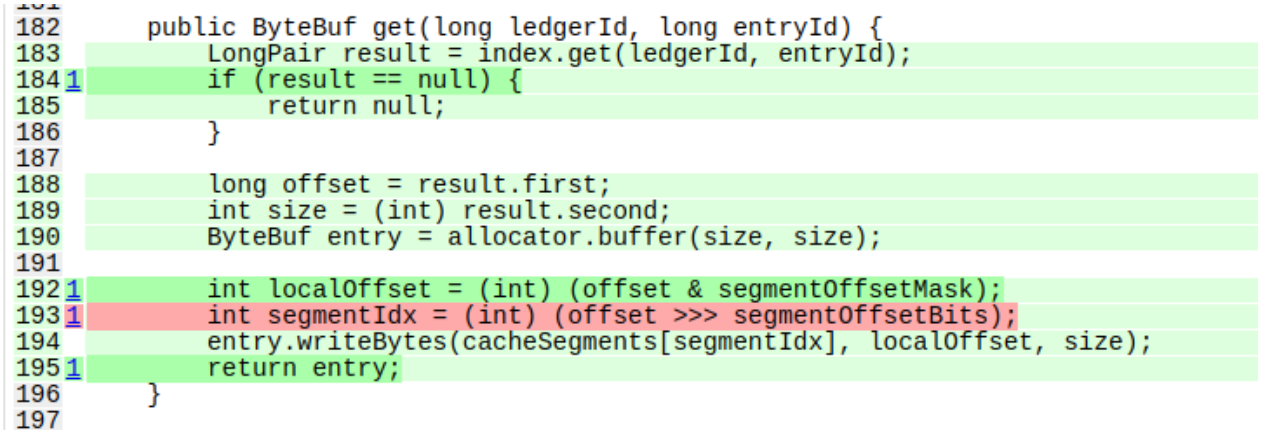
WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda$forEach$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)		0%		n/a	1	1	2	2	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		99%		87%	1	5	1	21	0	1
get(long, long)		100%		100%	0	2	0	10	0	1
clear()		100%		n/a	0	1	0	7	0	1
close()		100%		100%	0	2	0	3	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
count()		100%		n/a	0	1	0	1	0	1
Total	265 of 631	58%	23 of 38	39%	20	35	51	121	7	16

Figura 22: WriteCache Coverage dopo l'aggiunta di ulteriori test

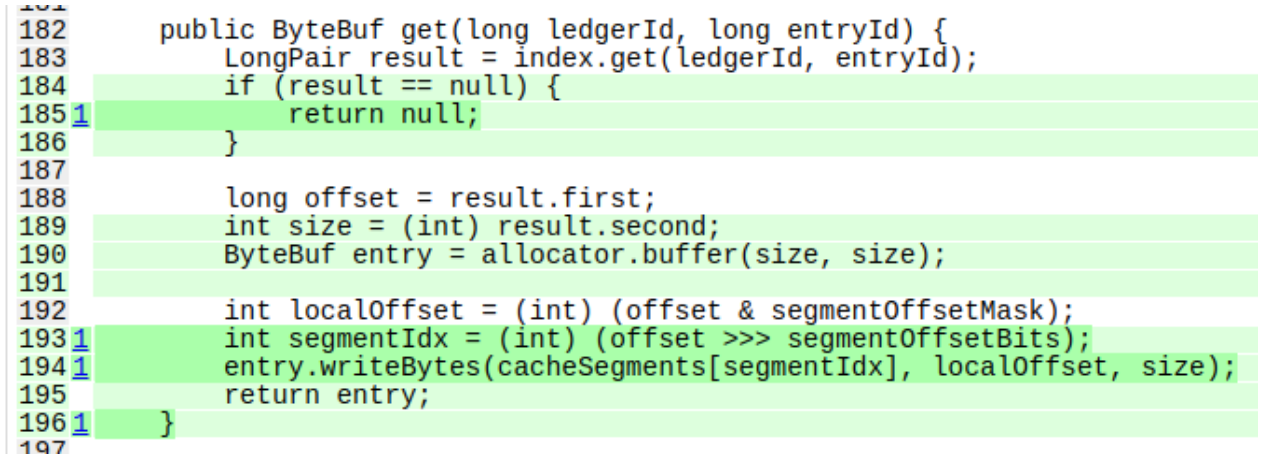
```
183.    public ByteBuf get(long ledgerId, long entryId) {
184.        LongPair result = index.get(ledgerId, entryId);
185.        if (result == null) {
186.            return null;
187.        }
188.
189.        long offset = result.first;
190.        int size = (int) result.second;
191.        ByteBuf entry = allocator.buffer(size, size);
192.
193.        int localOffset = (int) (offset & segmentOffsetMask);
194.        int segmentIdx = (int) (offset >>> segmentOffsetBits);
195.        entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196.        return entry;
197.    }
```

Figura 23: WriteCache ***get*** Coverage



```
182     public ByteBuf get(long ledgerId, long entryId) {
183         LongPair result = index.get(ledgerId, entryId);
184 1       if (result == null) {
185             return null;
186         }
187
188         long offset = result.first;
189         int size = (int) result.second;
190         ByteBuf entry = allocator.buffer(size, size);
191
192 1       int localOffset = (int) (offset & segmentOffsetMask);
193 1       int segmentIdx = (int) (offset >>> segmentOffsetBits);
194         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
195 1       return entry;
196     }
197
```

Figura 24: WriteCache ***get*** Mutation Suite Minimale

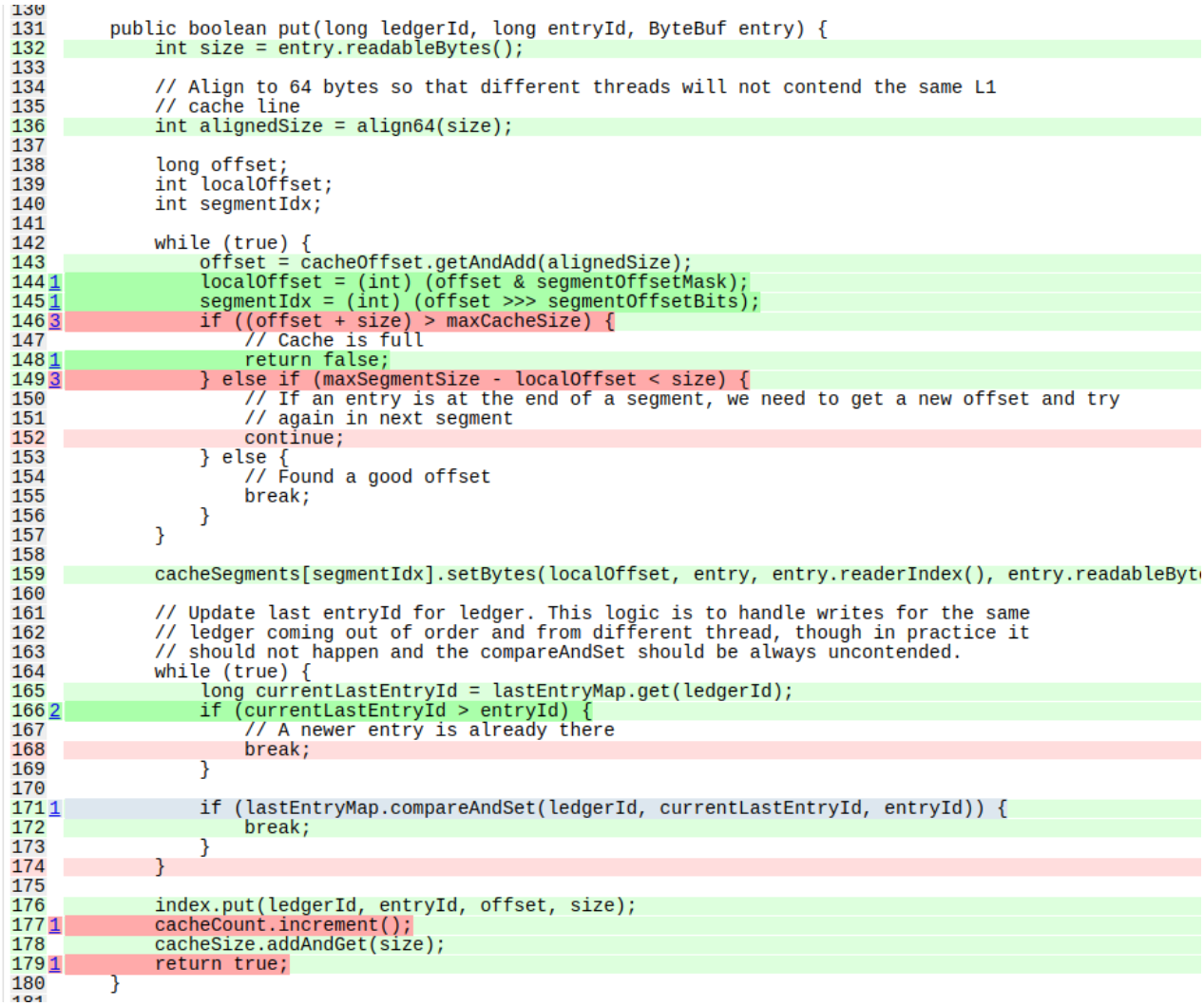


```
182     public ByteBuf get(long ledgerId, long entryId) {
183         LongPair result = index.get(ledgerId, entryId);
184         if (result == null) {
185 1           return null;
186         }
187
188         long offset = result.first;
189         int size = (int) result.second;
190         ByteBuf entry = allocator.buffer(size, size);
191
192         int localOffset = (int) (offset & segmentOffsetMask);
193 1       int segmentIdx = (int) (offset >>> segmentOffsetBits);
194 1       entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
195         return entry;
196 1   }
197
```

Figura 25: WriteCache ***get*** Mutation dopo l'aggiunta di ulteriori test

```
131.    public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132.        int size = entry.readableBytes();
133.
134.        // Align to 64 bytes so that different threads will not contend the same L1
135.        // cache line
136.        int alignedSize = align64(size);
137.
138.        long offset;
139.        int localOffset;
140.        int segmentIdx;
141.
142.        while (true) {
143.            offset = cacheOffset.getAndAdd(alignedSize);
144.            localOffset = (int) (offset & segmentOffsetMask);
145.            segmentIdx = (int) (offset >>> segmentOffsetBits);
146.            if ((offset + size) > maxCacheSize) {
147.                // Cache is full
148.                return false;
149.            } else if (maxSegmentSize - localOffset < size) {
150.                // If an entry is at the end of a segment, we need to get a new offset and try
151.                // again in next segment
152.                continue;
153.            } else {
154.                // Found a good offset
155.                break;
156.            }
157.        }
158.
159.        cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
160.
161.        // Update last entryId for ledger. This logic is to handle writes for the same
162.        // ledger coming out of order and from different thread, though in practice it
163.        // should not happen and the compareAndSet should be always uncontended.
164.        while (true) {
165.            long currentLastEntryId = lastEntryMap.get(ledgerId);
166.            System.out.println("CURRENT "+currentLastEntryId+" ENTRYID "+entryId);
167.            if (currentLastEntryId > entryId) {
168.                // A newer entry is already there
169.                break;
170.            }
171.
172.            if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173.                break;
174.            }
175.        }
176.
177.        index.put(ledgerId, entryId, offset, size);
178.        cacheCount.increment();
179.        cacheSize.addAndGet(size);
180.        return true;
181.    }
```

Figura 26: WriteCache ***put*** Coverage



```
130
131     public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132         int size = entry.readableBytes();
133
134         // Align to 64 bytes so that different threads will not contend the same L1
135         // cache line
136         int alignedSize = align64(size);
137
138         long offset;
139         int localOffset;
140         int segmentIdx;
141
142         while (true) {
143             offset = cacheOffset.getAndAdd(alignedSize);
144 1           localOffset = (int) (offset & segmentOffsetMask);
145 1           segmentIdx = (int) (offset >>> segmentOffsetBits);
146 3           if ((offset + size) > maxCacheSize) {
147                 // Cache is full
148 1               return false;
149 3           } else if (maxSegmentSize - localOffset < size) {
150                 // If an entry is at the end of a segment, we need to get a new offset and try
151                 // again in next segment
152                 continue;
153             } else {
154                 // Found a good offset
155                 break;
156             }
157         }
158
159         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableByt
160
161         // Update last entryId for ledger. This logic is to handle writes for the same
162         // ledger coming out of order and from different thread, though in practice it
163         // should not happen and the compareAndSet should be always uncontended.
164         while (true) {
165             long currentLastEntryId = lastEntryMap.get(ledgerId);
166 2           if (currentLastEntryId > entryId) {
167                 // A newer entry is already there
168                 break;
169             }
170
171 1           if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
172                 break;
173             }
174         }
175
176         index.put(ledgerId, entryId, offset, size);
177 1       cacheCount.increment();
178         cacheSize.addAndGet(size);
179 1       return true;
180     }
```

Figura 27: WriteCache ***put*** Mutation Suite Minimale

```
131     public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132         int size = entry.readableBytes();
133
134         // Align to 64 bytes so that different threads will not contend the same L1
135         // cache line
136         int alignedSize = align64(size);
137
138         long offset;
139         int localOffset;
140         int segmentIdx;
141
142         while (true) {
143             offset = cacheOffset.getAndAdd(alignedSize);
144 1           localOffset = (int) (offset & segmentOffsetMask);
145 1           segmentIdx = (int) (offset >>> segmentOffsetBits);
146 3           if ((offset + size) > maxCacheSize) {
147                 // Cache is full
148 1               return false;
149 3           } else if (maxSegmentSize - localOffset < size) {
150                 // If an entry is at the end of a segment, we need to get a new offset and try
151                 // again in next segment
152                 continue;
153             } else {
154                 // Found a good offset
155                 break;
156             }
157         }
158
159         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
160
161         // Update last entryId for ledger. This logic is to handle writes for the same
162         // ledger coming out of order and from different thread, though in practice it
163         // should not happen and the compareAndSet should be always uncontended.
164         while (true) {
165             long currentLastEntryId = lastEntryMap.get(ledgerId);
166 1           if (currentLastEntryId > entryId) {
167 2               // A newer entry is already there
168                 break;
169             }
170
171             if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
172 1               break;
173             }
174         }
175
176         index.put(ledgerId, entryId, offset, size);
177         cacheCount.increment();
178 1       cacheSize.addAndGet(size);
179         return true;
180 1   }
181
```

Figura 28: WriteCache ***put*** Mutation dopo l'aggiunta di ulteriori test

ZKUtil

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
deleteInBatch(ZooKeeper, List, int)		0%		0%	5	5	14	14	1	1
deleteRecursive(ZooKeeper, String, AsyncCallback.VoidCallback, Object)		0%		0%	2	2	6	6	1	1
deleteRecursive(ZooKeeper, String, int)		0%		n/a	1	1	4	4	1	1
lambda$deleteInBatch$0(int, String, Object, List)		0%		0%	2	2	4	4	1	1
deleteRecursive(ZooKeeper, String)		0%		n/a	1	1	2	2	1	1
ZKUtil()		0%		n/a	1	1	1	1	1	1
visitSubTreeDFSHelper(ZooKeeper, String, boolean, AsyncCallback.StringCallback)		97%		100%	0	6	2	15	0	1
listSubTreeBFS(ZooKeeper, String)		100%		100%	0	4	0	14	0	1
validateFileInput(String)		100%		100%	0	4	0	8	0	1
constructPermString(int)		100%		100%	0	6	0	12	0	1
aclToString(List)		100%		100%	0	2	0	9	0	1
visitSubTreeDFS(ZooKeeper, String, boolean, AsyncCallback.StringCallback)		100%		n/a	0	1	0	5	0	1
static {...}		100%		n/a	0	1	0	2	0	1
getPermString(int)		100%		n/a	0	1	0	1	0	1
lambda$getPermString$1(Integer)		100%		n/a	0	1	0	1	0	1
Total	131 of 473	72%	12 of 46	73%	12	38	33	97	6	15

Figura 29: ZKUtil Coverage

```
171.    /**
172.     * BFS Traversal of the system under pathRoot, with the entries in the list, in the
173.     * same order as that of the traversal.
174.     * <p>
175.     * <b>Important:</b> This is <i>not an atomic snapshot</i> of the tree ever, but the
176.     *  state as it exists across multiple RPCs from zkClient to the ensemble.
177.     * For practical purposes, it is suggested to bring the clients to the ensemble
178.     * down (i.e. prevent writes to pathRoot) to 'simulate' a snapshot behavior.
179.     *
180.     * @param zk the zookeeper handle
181.     * @param pathRoot The znode path, for which the entire subtree needs to be listed.
182.     * @throws InterruptedException
183.     * @throws KeeperException
184.     */
185.    public static List<String> listSubTreeBFS(
186.        ZooKeeper zk,
187.        final String pathRoot) throws KeeperException, InterruptedException {
188.        Queue<String> queue = new ArrayDeque<>();
189.        List<String> tree = new ArrayList<String>();
190.        queue.add(pathRoot);
191.        tree.add(pathRoot);
192.        while (!queue.isEmpty()) {
193.            String node = queue.poll();
194.            List<String> children = zk.getChildren(node, false);
195.            for (final String child : children) {
196.                // Fix IllegalArgumentException when list "/".
197.                final String childPath = (node.equals("/") ? "" : node) + "/" + child;
198.                queue.add(childPath);
199.                tree.add(childPath);
200.            }
201.        }
202.        return tree;
203.    }
```

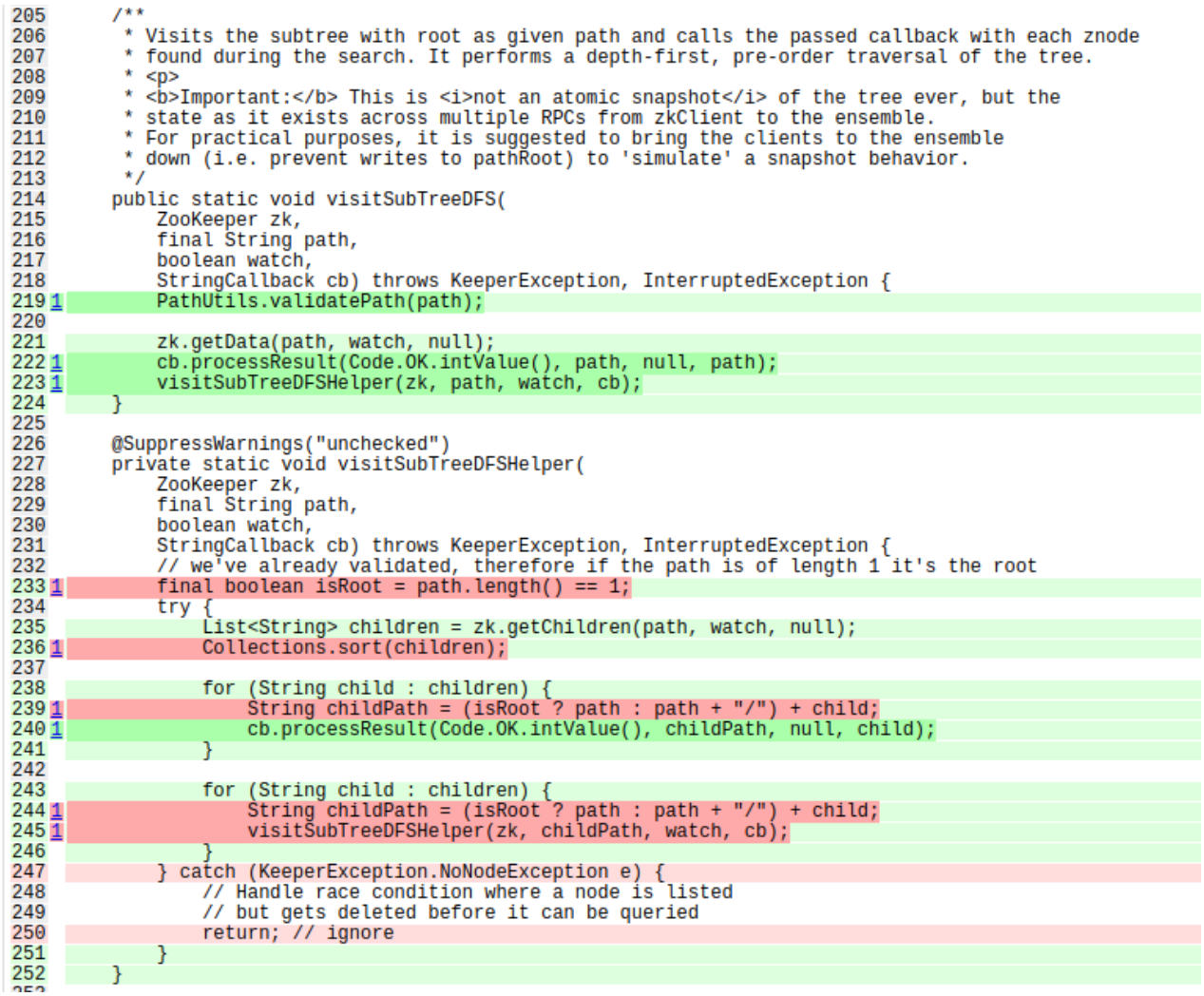
Figura 30: ZKUtil ***listSubTreeBFS*** Coverage

```
171     /**
172      * BFS Traversal of the system under pathRoot, with the entries in the list, in the
173      * same order as that of the traversal.
174      * <p>
175      * <b>Important:</b> This is <i>not an atomic snapshot</i> of the tree ever, but the
176      *  state as it exists across multiple RPCs from zkClient to the ensemble.
177      * For practical purposes, it is suggested to bring the clients to the ensemble
178      * down (i.e. prevent writes to pathRoot) to 'simulate' a snapshot behavior.
179      *
180      * @param zk the zookeeper handle
181      * @param pathRoot The znode path, for which the entire subtree needs to be listed.
182      * @throws InterruptedException
183      * @throws KeeperException
184      */
185     public static List<String> listSubTreeBFS(
186         ZooKeeper zk,
187         final String pathRoot) throws KeeperException, InterruptedException {
188         Queue<String> queue = new ArrayDeque<>();
189         List<String> tree = new ArrayList<String>();
190         queue.add(pathRoot);
191         tree.add(pathRoot);
1921        while (!queue.isEmpty()) {
193             String node = queue.poll();
194             List<String> children = zk.getChildren(node, false);
195             for (final String child : children) {
196                 // Fix IllegalArgumentException when list "/".
1971                final String childPath = (node.equals("/") ? "" : node) + "/" + child;
198                 queue.add(childPath);
199                 tree.add(childPath);
200             }
201         }
2021        return tree;
203     }
```

Figura 31: ZKUtil ***listSubTreeBFS*** Mutation

```
207.      * found during the search. It performs a depth-first, pre-order traversal of the tre
208.      * <p>
209.      * <b>Important:</b> This is <i>not an atomic snapshot</i> of the tree ever, but the
210.      * state as it exists across multiple RPCs from zkClient to the ensemble.
211.      * For practical purposes, it is suggested to bring the clients to the ensemble
212.      * down (i.e. prevent writes to pathRoot) to 'simulate' a snapshot behavior.
213.      */
214.     public static void visitSubTreeDFS(
215.         ZooKeeper zk,
216.         final String path,
217.         boolean watch,
218.         StringCallback cb) throws KeeperException, InterruptedException {
219.         PathUtils.validatePath(path);
220.
221.         zk.getData(path, watch, null);
222.         cb.processResult(Code.OK.intValue(), path, null, path);
223.         visitSubTreeDFSHelper(zk, path, watch, cb);
224.     }
225.
226.     @SuppressWarnings("unchecked")
227.     private static void visitSubTreeDFSHelper(
228.         ZooKeeper zk,
229.         final String path,
230.         boolean watch,
231.         StringCallback cb) throws KeeperException, InterruptedException {
232.         // we've already validated, therefore if the path is of length 1 it's the root
233.         final boolean isRoot = path.length() == 1;
234.         try {
235.             List<String> children = zk.getChildren(path, watch, null);
236.             Collections.sort(children);
237.
238.             for (String child : children) {
239.                 String childPath = (isRoot ? path : path + "/") + child;
240.                 cb.processResult(Code.OK.intValue(), childPath, null, child);
241.             }
242.
243.             for (String child : children) {
244.                 String childPath = (isRoot ? path : path + "/") + child;
245.                 visitSubTreeDFSHelper(zk, childPath, watch, cb);
246.             }
247.         } catch (KeeperException.NoNodeException e) {
248.             // Handle race condition where a node is listed
249.             // but gets deleted before it can be queried
250.             return; // ignore
251.         }
252.     }
```

Figura 32: ZKUtil ***visitSubTreeDFS*** Coverage



```
205      /**
206       * Visits the subtree with root as given path and calls the passed callback with each znode
207       * found during the search. It performs a depth-first, pre-order traversal of the tree.
208       * <p>
209       * <b>Important:</b> This is <i>not an atomic snapshot</i> of the tree ever, but the
210       * state as it exists across multiple RPCs from zkClient to the ensemble.
211       * For practical purposes, it is suggested to bring the clients to the ensemble
212       * down (i.e. prevent writes to pathRoot) to 'simulate' a snapshot behavior.
213       */
214      public static void visitSubTreeDFS(
215          ZooKeeper zk,
216          final String path,
217          boolean watch,
218          StringCallback cb) throws KeeperException, InterruptedException {
219 1        PathUtils.validatePath(path);
220
221          zk.getData(path, watch, null);
222 1        cb.processResult(Code.OK.intValue(), path, null, path);
223 1        visitSubTreeDFSHelper(zk, path, watch, cb);
224      }
225
226      @SuppressWarnings("unchecked")
227      private static void visitSubTreeDFSHelper(
228          ZooKeeper zk,
229          final String path,
230          boolean watch,
231          StringCallback cb) throws KeeperException, InterruptedException {
232          // we've already validated, therefore if the path is of length 1 it's the root
233 1        final boolean isRoot = path.length() == 1;
234          try {
235              List<String> children = zk.getChildren(path, watch, null);
236 1            Collections.sort(children);
237
238              for (String child : children) {
239 1                String childPath = (isRoot ? path : path + "/") + child;
240 1                cb.processResult(Code.OK.intValue(), childPath, null, child);
241              }
242
243              for (String child : children) {
244 1                String childPath = (isRoot ? path : path + "/") + child;
245 1                visitSubTreeDFSHelper(zk, childPath, watch, cb);
246              }
247          } catch (KeeperException.NoNodeException e) {
248              // Handle race condition where a node is listed
249              // but gets deleted before it can be queried
250              return; // ignore
251          }
252      }
```

Figura 33: ZKUtil ***visitSubTreeDFS*** Mutation Suite Minimale

```
205      /**
206       * Visits the subtree with root as given path and calls the passed callback with each znode
207       * found during the search. It performs a depth-first, pre-order traversal of the tree.
208       * <p>
209       * <b>Important:</b> This is <i>not an atomic snapshot</i> of the tree ever, but the
210       * state as it exists across multiple RPCs from zkClient to the ensemble.
211       * For practical purposes, it is suggested to bring the clients to the ensemble
212       * down (i.e. prevent writes to pathRoot) to 'simulate' a snapshot behavior.
213       */
214      public static void visitSubTreeDFS(
215          ZooKeeper zk,
216          final String path,
217          boolean watch,
218          StringCallback cb) throws KeeperException, InterruptedException {
219 1        PathUtils.validatePath(path);
220
221          zk.getData(path, watch, null);
222 1        cb.processResult(Code.OK.intValue(), path, null, path);
223 1        visitSubTreeDFSHelper(zk, path, watch, cb);
224      }
225
226      @SuppressWarnings("unchecked")
227      private static void visitSubTreeDFSHelper(
228          ZooKeeper zk,
229          final String path,
230          boolean watch,
231          StringCallback cb) throws KeeperException, InterruptedException {
232          // we've already validated, therefore if the path is of length 1 it's the root
233 1        final boolean isRoot = path.length() == 1;
234          try {
235              List<String> children = zk.getChildren(path, watch, null);
236 1            Collections.sort(children);
237
238              for (String child : children) {
239 1                String childPath = (isRoot ? path : path + "/") + child;
240 1                cb.processResult(Code.OK.intValue(), childPath, null, child);
241              }
242
243              for (String child : children) {
244 1                String childPath = (isRoot ? path : path + "/") + child;
245 1                visitSubTreeDFSHelper(zk, childPath, watch, cb);
246              }
247          } catch (KeeperException.NoNodeException e) {
248              // Handle race condition where a node is listed
249              // but gets deleted before it can be queried
250              return; // ignore
251          }
252      }
```

Figura 34: ZKUtil ***visitSubTreeDFS*** Mutation dopo l'aggiunta di ulteriori test

```
153.     /**
154.      * @param filePath the file path to be validated
155.      * @return Returns null if valid otherwise error message
156.      */
157.     public static String validateFileInput(String filePath) {
158.         File file = new File(filePath);
159.         if (!file.exists()) {
160.             return "File '" + file.getAbsolutePath() + "' does not exist.";
161.         }
162.         if (!file.canRead()) {
163.             return "Read permission is denied on the file '" + file.getAbsolutePath() + "'";
164.         }
165.         if (file.isDirectory()) {
166.             return "'" + file.getAbsolutePath() + "' is a directory. it must be a file.";
167.         }
168.         return null;
169.     }
```

Figura 35: ZKUtil ***validateFileInput*** Coverage

```
155       * @return Returns null if valid otherwise error message
156       */
157      public static String validateFileInput(String filePath) {
158          File file = new File(filePath);
1591         if (!file.exists()) {
1601             return "File '" + file.getAbsolutePath() + "' does not exist.";
161          }
1621         if (!file.canRead()) {
1631             return "Read permission is denied on the file '" + file.getAbsolutePath() + "'";
164          }
1651         if (file.isDirectory()) {
1661             return "'" + file.getAbsolutePath() + "' is a directory. it must be a file.";
167          }
1681         return null;
169      }
```

Figura 36: ZKUtil ***validateFileInput*** Mutation

```
283.     public static String aclToString(List<ACL> acls) {
284.         StringBuilder sb = new StringBuilder();
285.         for (ACL acl : acls) {
286.             sb.append(acl.getId().getScheme());
287.             sb.append(":");
288.             sb.append(acl.getId().getId());
289.             sb.append(":");
290.             sb.append(getPermString(acl.getPerms()));
291.         }
292.         return sb.toString();
293.     }
294. }
```

Figura 37: ZKUtil ***aclToString*** Coverage

```
283      public static String aclToString(List<ACL> acls) {
284          StringBuilder sb = new StringBuilder();
285          for (ACL acl : acls) {
286              sb.append(acl.getId().getScheme());
287              sb.append(":");
288              sb.append(acl.getId().getId());
289              sb.append(":");
290              sb.append(getPermString(acl.getPerms()));
291          }
2921         return sb.toString();
293      }
294  }
```

Figura 38: ZKUtil ***aclToString*** Mutation

```
259.     public static String getPermString(int perms) {
260.         return permCache.computeIfAbsent(perms, k -> constructPermString(k));
261.     }
262.
263.     private static String constructPermString(int perms) {
264.         StringBuilder p = new StringBuilder();
265.         if ((perms & ZooDefs.Perms.CREATE) != 0) {
266.             p.append('c');
267.         }
268.         if ((perms & ZooDefs.Perms.DELETE) != 0) {
269.             p.append('d');
270.         }
271.         if ((perms & ZooDefs.Perms.READ) != 0) {
272.             p.append('r');
273.         }
274.         if ((perms & ZooDefs.Perms.WRITE) != 0) {
275.             p.append('w');
276.         }
277.         if ((perms & ZooDefs.Perms.ADMIN) != 0) {
278.             p.append('a');
279.         }
280.         return p.toString();
281.     }
```

Figura 39: ZKUtil ***getPermString*** Coverage

```
254      /**
255       * @param perms
256       *            ACL permissions
257       * @return string representation of permissions
258       */
259      public static String getPermString(int perms) {
260 2        return permCache.computeIfAbsent(perms, k -> constructPermString(k));
261      }
262
263      private static String constructPermString(int perms) {
264          StringBuilder p = new StringBuilder();
265 2        if ((perms & ZooDefs.Perms.CREATE) != 0) {
266              p.append('c');
267          }
268 2        if ((perms & ZooDefs.Perms.DELETE) != 0) {
269              p.append('d');
270          }
271 2        if ((perms & ZooDefs.Perms.READ) != 0) {
272              p.append('r');
273          }
274 2        if ((perms & ZooDefs.Perms.WRITE) != 0) {
275              p.append('w');
276          }
277 2        if ((perms & ZooDefs.Perms.ADMIN) != 0) {
278              p.append('a');
279          }
280 1        return p.toString();
281      }
```

Figura 40: ZKUtil ***getPermString*** Mutation

FileTxnLog

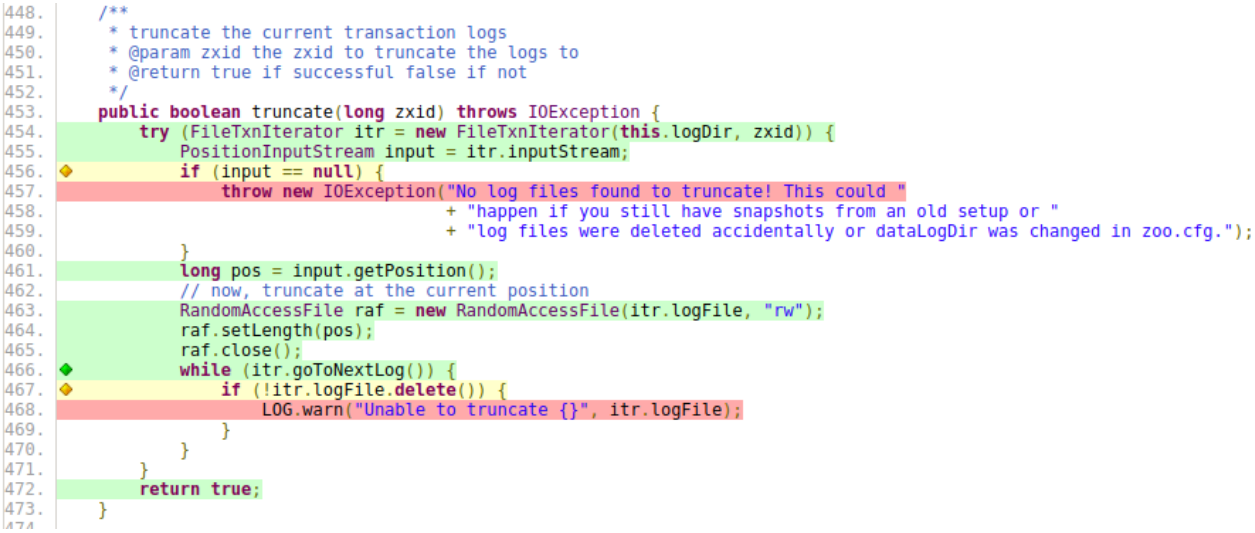
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getLastLoggedZxid()		0%		0%	3	3	13	13	1	1
commit()		54%		43%	7	9	11	26	0	1
readHeader(File)		0%		0%	2	2	11	11	1	1
getDbId()		0%		0%	2	2	6	6	1	1
rollLog()		0%		0%	2	2	6	6	1	1
static {...}		69%		50%	2	3	3	12	0	1
truncate(long)		76%		66%	2	4	2	12	0	1
getCurrentLogSize()		0%		0%	2	2	3	3	1	1
getTotalLogSize()		0%		n/a	1	1	1	1	1	1
append(TxnHeader, Record, TxnDigest)		96%		80%	2	6	1	28	0	1
setServerStats(ServerStats)		0%		n/a	1	1	2	2	1	1
setTotalLogSize(long)		0%		n/a	1	1	2	2	1	1
getLogFiles(File[], long)		95%		80%	2	6	2	17	0	1
setPreallocSize(long)		0%		n/a	1	1	2	2	1	1
setTxnLogSizeLimit(long)		0%		n/a	1	1	2	2	1	1
getTxnLogSyncElapsedTime()		0%		n/a	1	1	1	1	1	1
isForceSync()		0%		n/a	1	1	1	1	1	1
FileTxnLog(File)		97%		50%	1	2	0	10	0	1
close()		100%		100%	0	3	0	6	0	1
read(long, boolean)		100%		n/a	0	1	0	1	0	1
append(TxnHeader, Record)		100%		n/a	0	1	0	1	0	1
read(long)		100%		n/a	0	1	0	1	0	1
makeChecksumAlgorithm()		100%		n/a	0	1	0	1	0	1
Total	247 of 665	62%	30 of 64	53%	34	55	69	165	12	23

Figura 41: FileTxnLog Coverage Suite Minimale

FileTxnLog

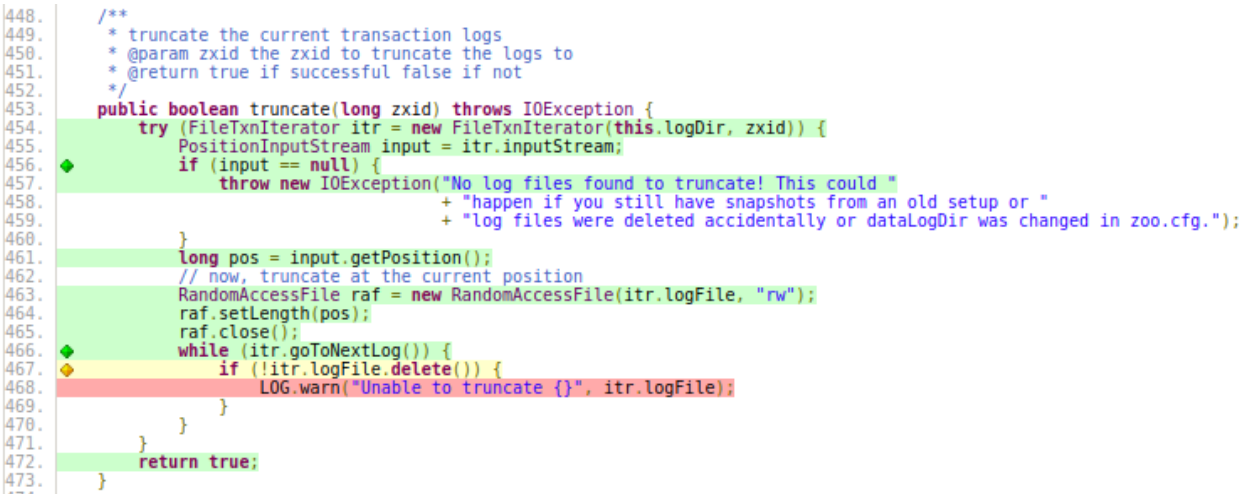
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getLastLoggedZxid()		0%		0%	3	3	13	13	1	1
commit()		54%		43%	7	9	11	26	0	1
readHeader(File)		0%		0%	2	2	11	11	1	1
getDbId()		0%		0%	2	2	6	6	1	1
rollLog()		0%		0%	2	2	6	6	1	1
static {...}		69%		50%	2	3	3	12	0	1
getCurrentLogSize()		0%		0%	2	2	3	3	1	1
truncate(long)		87%		83%	1	4	1	12	0	1
getTotalLogSize()		0%		n/a	1	1	1	1	1	1
append(TxnHeader, Record, TxnDigest)		96%		80%	2	6	1	28	0	1
setServerStats(ServerStats)		0%		n/a	1	1	2	2	1	1
setTotalLogSize(long)		0%		n/a	1	1	2	2	1	1
getLogFiles(File[], long)		95%		80%	2	6	2	17	0	1
setPreallocSize(long)		0%		n/a	1	1	2	2	1	1
setTxnLogSizeLimit(long)		0%		n/a	1	1	2	2	1	1
getTxnLogSyncElapsedTime()		0%		n/a	1	1	1	1	1	1
isForceSync()		0%		n/a	1	1	1	1	1	1
FileTxnLog(File)		97%		50%	1	2	0	10	0	1
close()		100%		100%	0	3	0	6	0	1
read(long, boolean)		100%		n/a	0	1	0	1	0	1
append(TxnHeader, Record)		100%		n/a	0	1	0	1	0	1
read(long)		100%		n/a	0	1	0	1	0	1
makeChecksumAlgorithm()		100%		n/a	0	1	0	1	0	1
Total	242 of 665	63%	29 of 64	54%	33	55	68	165	12	23

Figura 42: FileTxnLog Coverage dopo l'aggiunta di ulteriori test



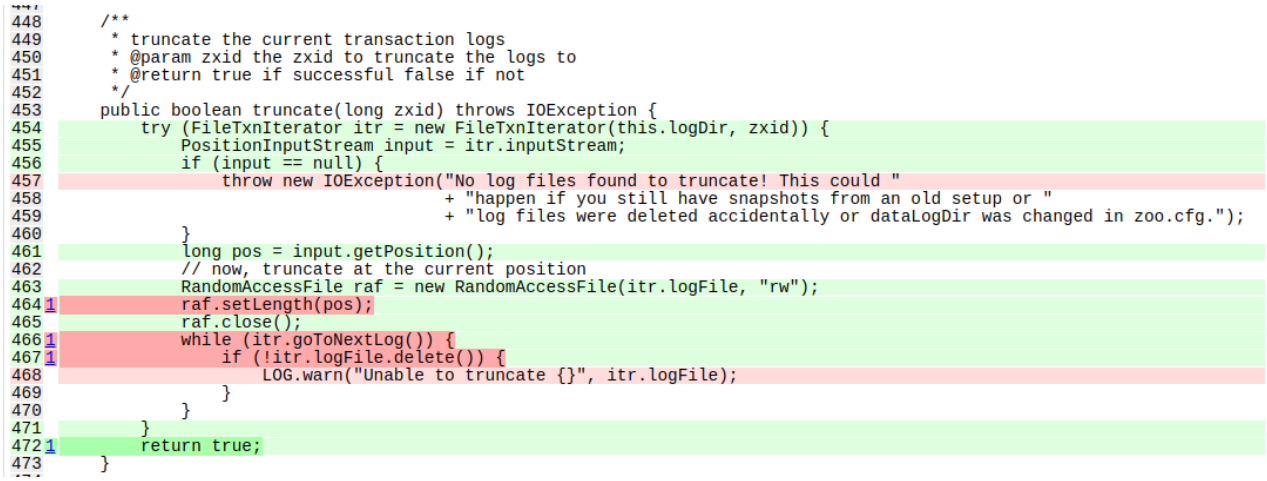
```
448.    /**
449.     * truncate the current transaction logs
450.     * @param zxid the zxid to truncate the logs to
451.     * @return true if successful false if not
452.     */
453.    public boolean truncate(long zxid) throws IOException {
454.        try (FileTxnIterator itr = new FileTxnIterator(this.logDir, zxid)) {
455.            PositionInputStream input = itr.inputStream;
456.            if (input == null) {
457.                throw new IOException("No log files found to truncate! This could "
458.                                      + "happen if you still have snapshots from an old setup or "
459.                                      + "log files were deleted accidentally or dataLogDir was changed in zoo.cfg.");
460.            }
461.            long pos = input.getPosition();
462.            // now, truncate at the current position
463.            RandomAccessFile raf = new RandomAccessFile(itr.logFile, "rw");
464.            raf.setLength(pos);
465.            raf.close();
466.            while (itr.goToNextLog()) {
467.                if (!itr.logFile.delete()) {
468.                    LOG.warn("Unable to truncate {}", itr.logFile);
469.                }
470.            }
471.        }
472.        return true;
473.    }
```

Figura 43: FileTxnLog ***truncate*** Coverage Suite Minimale



```
448.    /**
449.     * truncate the current transaction logs
450.     * @param zxid the zxid to truncate the logs to
451.     * @return true if successful false if not
452.     */
453.    public boolean truncate(long zxid) throws IOException {
454.        try (FileTxnIterator itr = new FileTxnIterator(this.logDir, zxid)) {
455.            PositionInputStream input = itr.inputStream;
456.            if (input == null) {
457.                throw new IOException("No log files found to truncate! This could "
458.                                      + "happen if you still have snapshots from an old setup or "
459.                                      + "log files were deleted accidentally or dataLogDir was changed in zoo.cfg.");
460.            }
461.            long pos = input.getPosition();
462.            // now, truncate at the current position
463.            RandomAccessFile raf = new RandomAccessFile(itr.logFile, "rw");
464.            raf.setLength(pos);
465.            raf.close();
466.            while (itr.goToNextLog()) {
467.                if (!itr.logFile.delete()) {
468.                    LOG.warn("Unable to truncate {}", itr.logFile);
469.                }
470.            }
471.        }
472.        return true;
473.    }
```

Figura 44: FileTxnLog ***truncate*** Coverage dopo l'aggiunta di ulteriori test



```
448     /**
449      * truncate the current transaction logs
450      * @param zxid the zxid to truncate the logs to
451      * @return true if successful false if not
452      */
453     public boolean truncate(long zxid) throws IOException {
454         try (FileTxnIterator itr = new FileTxnIterator(this.logDir, zxid)) {
455             PositionInputStream input = itr.inputStream;
456             if (input == null) {
457                 throw new IOException("No log files found to truncate! This could "
458                                       + "happen if you still have snapshots from an old setup or "
459                                       + "log files were deleted accidentally or dataLogDir was changed in zoo.cfg.");
460             }
461             long pos = input.getPosition();
462             // now, truncate at the current position
463             RandomAccessFile raf = new RandomAccessFile(itr.logFile, "rw");
464 1           raf.setLength(pos);
465             raf.close();
466 1           while (itr.goToNextLog()) {
467 1               if (!itr.logFile.delete()) {
468                     LOG.warn("Unable to truncate {}", itr.logFile);
469                 }
470             }
471         }
472 1       return true;
473     }
```

Figura 45: FileTxnLog ***truncate*** Mutation

```
431.    public TxnIterator read(long zxid) throws IOException {
432.        return read(zxid, true);
433.    }
434.
435.    /**
436.     * start reading all the transactions from the given zxid.
437.     *
438.     * @param zxid the zxid to start reading transactions from
439.     * @param fastForward true if the iterator should be fast forwarded to point
440.     *        to the txn of a given zxid, else the iterator will point to the
441.     *        starting txn of a txnlog that may contain txn of a given zxid
442.     * @return returns an iterator to iterate through the transaction logs
443.     */
444.    public TxnIterator read(long zxid, boolean fastForward) throws IOException {
445.        return new FileTxnIterator(logDir, zxid, fastForward);
446.    }
```

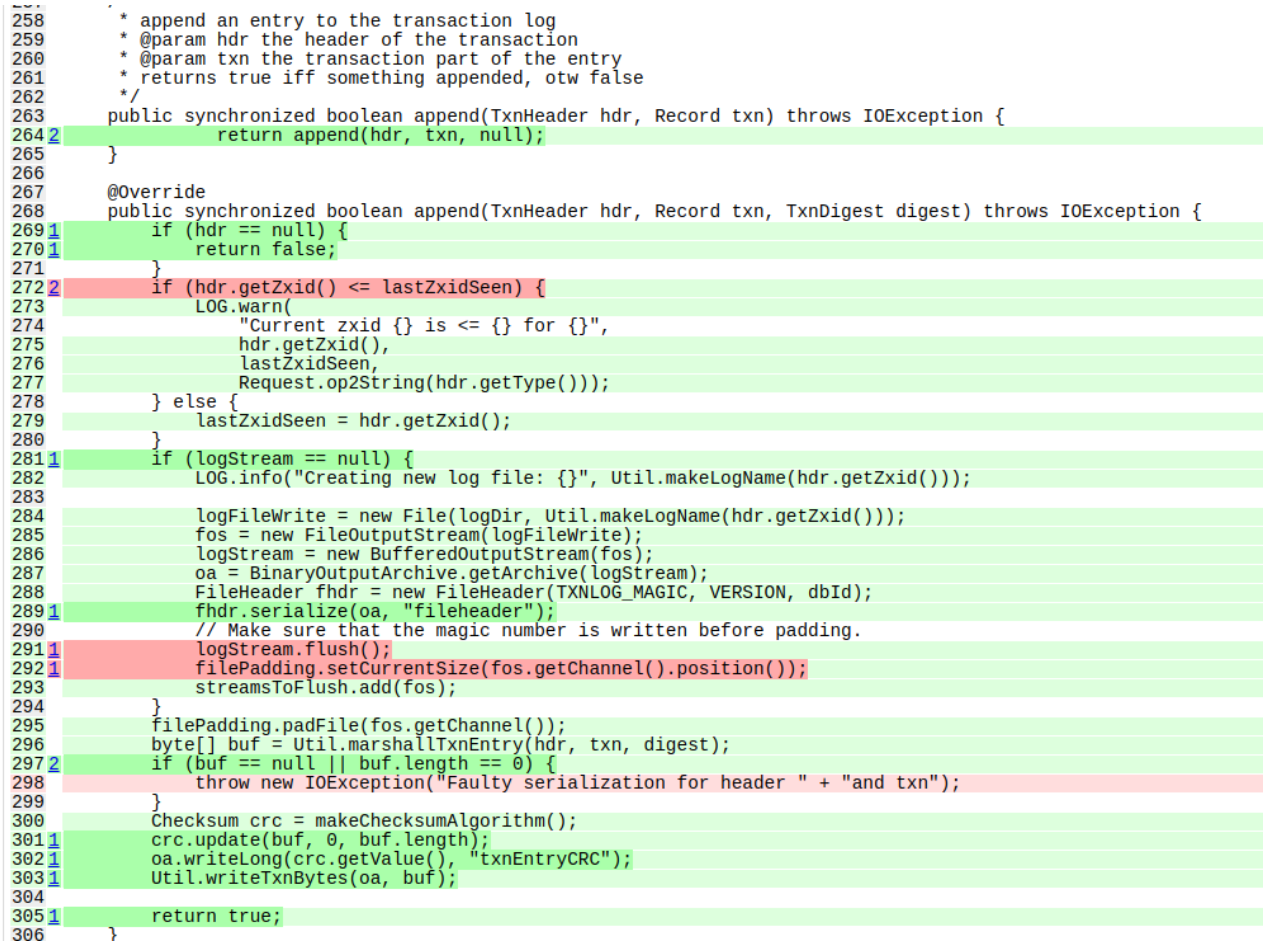
Figura 46: FileTxnLog ***read*** Coverage

```
425      /**
426       * start reading all the transactions from the given zxid
427       * @param zxid the zxid to start reading transactions from
428       * @return returns an iterator to iterate through the transaction
429       * logs
430       */
431      public TxnIterator read(long zxid) throws IOException {
432 1        return read(zxid, true);
433      }
434
435      /**
436       * start reading all the transactions from the given zxid.
437       *
438       * @param zxid the zxid to start reading transactions from
439       * @param fastForward true if the iterator should be fast forwarded to point
440       *        to the txn of a given zxid, else the iterator will point to the
441       *        starting txn of a txnlog that may contain txn of a given zxid
442       * @return returns an iterator to iterate through the transaction logs
443       */
444      public TxnIterator read(long zxid, boolean fastForward) throws IOException {
445 1        return new FileTxnIterator(logDir, zxid, fastForward);
446      }
```

Figura 47: FileTxnLog ***read*** Mutation

```
267.    @Override
268.    public synchronized boolean append(TxnHeader hdr, Record txn, TxnDigest digest) throws IOException {
269.        if (hdr == null) {
270.            return false;
271.        }
272.        if (hdr.getZxid() <= lastZxidSeen) {
273.            LOG.warn(
274.                "Current zxid {} is <= {} for {}",
275.                hdr.getZxid(),
276.                lastZxidSeen,
277.                Request.op2String(hdr.getType()));
278.        } else {
279.            lastZxidSeen = hdr.getZxid();
280.        }
281.        if (logStream == null) {
282.            LOG.info("Creating new log file: {}", Util.makeLogName(hdr.getZxid()));
283.
284.            logFileWrite = new File(logDir, Util.makeLogName(hdr.getZxid()));
285.            fos = new FileOutputStream(logFileWrite);
286.            logStream = new BufferedOutputStream(fos);
287.            oa = BinaryOutputArchive.getArchive(logStream);
288.            FileHeader fhdr = new FileHeader(TXNLOG_MAGIC, VERSION, dbId);
289.            fhdr.serialize(oa, "fileheader");
290.            // Make sure that the magic number is written before padding.
291.            logStream.flush();
292.            filePadding.setCurrentSize(fos.getChannel().position());
293.            streamsToFlush.add(fos);
294.        }
295.        filePadding.padFile(fos.getChannel());
296.        byte[] buf = Util.marshallTxnEntry(hdr, txn, digest);
297.        if (buf == null || buf.length == 0) {
298.            throw new IOException("Faulty serialization for header " + "and txn");
299.        }
300.        Checksum crc = makeChecksumAlgorithm();
301.        crc.update(buf, 0, buf.length);
302.        oa.writeLong(crc.getValue(), "txnEntryCRC");
303.        Util.writeTxnBytes(oa, buf);
304.
305.        return true;
306.    }
```

Figura 48: FileTxnLog ***append*** Coverage



```
258       * append an entry to the transaction log
259       * @param hdr the header of the transaction
260       * @param txn the transaction part of the entry
261       * returns true iff something appended, otw false
262       */
263      public synchronized boolean append(TxnHeader hdr, Record txn) throws IOException {
264 2              return append(hdr, txn, null);
265      }
266
267      @Override
268      public synchronized boolean append(TxnHeader hdr, Record txn, TxnDigest digest) throws IOException {
269 1        if (hdr == null) {
270 1            return false;
271          }
272 2        if (hdr.getZxid() <= lastZxidSeen) {
273              LOG.warn(
274                  "Current zxid {} is <= {} for {}",
275                  hdr.getZxid(),
276                  lastZxidSeen,
277                  Request.op2String(hdr.getType()));
278          } else {
279              lastZxidSeen = hdr.getZxid();
280          }
281 1        if (logStream == null) {
282              LOG.info("Creating new log file: {}", Util.makeLogName(hdr.getZxid()));
283
284              logFileWrite = new File(logDir, Util.makeLogName(hdr.getZxid()));
285              fos = new FileOutputStream(logFileWrite);
286              logStream = new BufferedOutputStream(fos);
287              oa = BinaryOutputArchive.getArchive(logStream);
288              FileHeader fhdr = new FileHeader(TXNLOG_MAGIC, VERSION, dbId);
289 1            fhdr.serialize(oa, "fileheader");
290              // Make sure that the magic number is written before padding.
291 1            logStream.flush();
292 1            filePadding.setCurrentSize(fos.getChannel().position());
293              streamsToFlush.add(fos);
294          }
295          filePadding.padFile(fos.getChannel());
296          byte[] buf = Util.marshallTxnEntry(hdr, txn, digest);
297 2        if (buf == null || buf.length == 0) {
298              throw new IOException("Faulty serialization for header " + "and txn");
299          }
300          Checksum crc = makeChecksumAlgorithm();
301 1        crc.update(buf, 0, buf.length);
302 1        oa.writeLong(crc.getValue(), "txnEntryCRC");
303 1        Util.writeTxnBytes(oa, buf);
304
305 1        return true;
306      }
```

Figura 49: FileTxnLog ***append*** Mutation