

Report ISW2 - Modulo Software Testing

Jacopo Fabi 0293870

SOMMARIO

Sommario	1
1 Introduzione	2
2 JCS	2
2.1 JCSUniTest.java	2
2.2 RemovalTestUtil.java	3
2.3 Coverage	3
3 Bookkeeper	6
3.1 ReadCache	6
3.1.1 public void put (long ledgerId, long entryId, ByteBuf entry)	6
3.1.2 public ByteBuf get(long ledgerId, long entryId)	8
3.2 DigestManager	8
3.2.1 public ByteBufList computeDigestAndPackageForSending(long entryId, long lastAddConfirmed, long length, ByteBuf data)	8
3.2.2 private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck) ...	10
4 Avro	12
4.1 BinaryData	12
4.1.1 private static int compare(Decoders d, Schema schema)	12
4.2 SpecificData	14
4.2.1 protected createSchema(final String realmPath, final String groupKey)	14
5 Link	16
5.1 JCS*	16
5.2 Bookkeeper	16
5.3 Avro	16
6 Appendice	17

1 INTRODUZIONE

L'obiettivo del report è quello di documentare e descrivere lo svolgimento delle attività di testing, motivando le scelte effettuate, e riportando i risultati ottenuti sui due progetti open-source analizzati, *Bookkeeper* e *Avro*.

Per l'implementazione dei casi di test sono stati utilizzati JUnit4 e Mockito, sfruttando Eclipse come IDE di sviluppo, mentre i framework JaCoCo e Pit permettono rispettivamente di generare i report sul control-flow coverage e di realizzare il mutation testing.

Le attività di building e testing sono state inserite in un contesto di Continuous Integration, così da poter analizzare, valutare ed eventualmente migliorare i risultati ottenuti ad ogni commit.

La piattaforma TravisCI permette di effettuare il build e il test del progetto in maniera automatica su un ambiente controllato differente da quello di sviluppo, mentre SonarCloud permette di analizzare il coverage ottenuto a seguito dell'esecuzione dei test sul progetto.

Il ciclo di build viene gestito tramite Maven, che a seguito di ogni commit:

1. Effettua il build del progetto in locale e su TravisCI
2. Attiva gli appositi profili definiti per JaCoCo e Pit, generando i report sul coverage e sul mutation testing
3. Carica le informazioni sul coverage sulla piattaforma SonarCloud

2 JCS

JCS (Java Caching System) è un sistema di cache distribuito scritto in Java. Lo scopo di questa attività è quello di scrivere dei test parametrici, andando a convertire da JUnit3 a JUnit4 i test originali di JCS.

Seguendo l'algoritmo di scelta, i test considerati sono quelli presenti all'interno delle classi

`JCSUniTest.java` e `RemovalTestUtil.java`.

Per convertire i test da JUnit3 a JUnit4 non è stata modificata l'implementazione dei test originali, ma ogni metodo della Test Suite è stato semplicemente annotato con `@Test`. Per rendere i test parametrici è stato utilizzato il runner `Parametrized.class`, per cui, di conseguenza, è stato definito un metodo `data()` con annotazione `@Parameters` per specificare i parametri da passare al costruttore della classe di test.

Successivamente sono stati ricavati i parametri utilizzati nei vari casi di test, in modo da poter inserirli nel metodo `data()` e costruire una collezione di parametri che permettono l'esecuzione dei diversi test case con valori differenti per i parametri considerati.

Questa novità di JUnit4 rende molto più semplice la gestione e l'ampliamento della test suite, in quanto per descrivere un nuovo caso di test basterà aggiungere una nuova entry nella lista restituita dal metodo `data()`.

Il setup dell'ambiente è stato definito all'interno del metodo `configure()` con annotazione `@BeforeClass`. Questo metodo, che viene eseguito una sola volta prima dell'esecuzione dei casi di test, si occupa di ottenere un'istanza di JCS e di impostare il corretto file `.ccf` di configurazione.

In linea generale, per riscrivere i nuovi Test Case si è cercato di utilizzare un approccio di tipo black-box, basandosi sui commenti presenti nel codice e nei casi di test originali.

2.1 JCSUNITEST.JAVA

L'obiettivo di questa classe è quello di effettuare un semplice test sulla classe JCS, verificando il corretto inserimento di un oggetto all'interno di un'istanza di JCS, che accede alla regione specificata. In particolare, viene invocato `getInstance()` per accedere ad una precisa istanza di JCS, si effettua il `put()` per inserire una entry in cache, e successivamente si va a verificare che l'entry è stata inserita correttamente sfruttando il metodo `get()`.

Durante la progettazione del test parametrico sono stati identificati come valori variabili i parametri del metodo `put()`, che sono l'oggetto da inserire in cache, `obj`, e la chiave che gli si va ad assegnare, `name`. La classe `JCSUnitTest.java` presenta un unico test case all'interno, per cui `obj` e `name` vengono testati una sola volta con un singolo valore, e quindi per una prima valutazione della copertura strutturale sono stati utilizzati soltanto i valori predefiniti come input.

2.2 REMOVALTESTUTIL.JAVA

L'obiettivo di questa classe è quello di testare l'inserimento, l'ottenimento, e la rimozione di un insieme di entry nel range specificato, tramite i metodi `put()`, `get()` e `remove()`.

I parametri individuati per la progettazione del test parametrico sono l'indice iniziale e finale del range usato per inserire, recuperare e rimuovere le entry, che sono proprio i parametri dei test originali. Il metodo di test `runGetInRange()`, oltre ai parametri `start` e `end` utilizza un ulteriore parametro booleano `check` per stabilire se un entry nulla è accettabile o meno. Per una prima valutazione sul coverage dei test originali sono stati utilizzati soltanto i valori di default trovati nella classe `ConcurrentRemovalLoadTest.java`, che sono rispettivamente:

- o `RunTestPutThenRemoveCategorical(): {100,200} {601,700} {701,800}`
- o `RunPutInRange(): {300,400} {401,600}`
- o `RunGetInRange(): {0,1000,false}`

Essendo il test parametrico, ogni entry della lista di parametri dovrà presentare tutti i parametri necessari all'esecuzione dei metodi di test presenti. Per questo motivo, ogni entry presenterà, oltre ai parametri `start` e `end`, anche il parametro booleano `check`, necessario per l'esecuzione del metodo `RunGetInRange()` ma non per eseguire gli altri metodi.

Il numero di parametri viene ampliato per evitare che l'esecuzione dei metodi `RunPutInRange()` e `RunTestPutThenRemoveCategorical()` fallisca con i parametri `start=0` e `end=1000` del metodo `RunGetInRange()`:

- Per ogni entry aggiungiamo un tipo, che identifica il preciso metodo di test
- Quando la classe viene istanziata, il tipo specificato permette a JUnit di capire, per ogni metodo di test, se deve essere eseguito o saltato, sfruttando `assumeTrue`

In questo modo eseguiamo ogni metodo di test solamente con i propri valori di default, andando anche ad evitare errori dovuti all'esecuzione di test con valori non previsti.

2.3 COVERAGE

Per analizzare il coverage ottenuto tramite le due classi di test è stato specificato all'interno del `pom.xml` il plugin per l'uso di `JaCoCo`. Durante il ciclo di build Maven, se viene attivato il plugin, verrà eseguito `JaCoCo` per stimare il coverage ottenuto dall'esecuzione dei casi di test. Non essendo presente il codice sorgente di JCS, `JaCoCo` non restituisce alcun report sul coverage, per cui è necessario utilizzare il plugin su una versione instrumentata di `jcs.jar`.

Al termine dell'esecuzione del goal `prepare-agent`, `JaCoCo` genera un report di tipo `.exec`, che può essere analizzato caricandolo su Eclipse, oppure deve essere convertito in un formato human-readable sfruttando uno script che converte l'`exec` in diversi formati, tra cui `html`.

Durante questa fase di analisi è stato utilizzato un approccio di tipo black-box, andando ad osservare il codice nello specifico in modo tale da motivare i risultati ottenuti dal report sul coverage. La classe stimolata dalle due classi di test considerate è `org.apache.jcs.access.CacheAccess.java` ed in particolare vengono sollecitati i metodi `get()`, `put()`, `remove()` usati per recuperare, inserire e rimuovere entry da un'istanza di JCS.

I risultati del coverage [Figura 1] mostrano che per i metodi `get()`, `put()` e `remove()` è stato raggiunto il 100% di coverage, ciò vuol dire che i casi di test originali di JCS vanno già a testare tutte le istruzioni presenti nelle implementazioni di questi metodi.

Tramite un approccio white-box, si evince che il metodo `put()` invoca semplicemente un altro metodo `put()` con 60% di coverage, ed è quest'ultimo che presenta il codice per l'inserimento di un entry. Ispezionando il codice del metodo, si è notato che il motivo per cui non si raggiunge il 100% di coverage è dovuto al fatto che la test suite di JCS non prevede nemmeno un caso di test che:

1. Inserisca una entry con chiave nulla
2. Inserisca una entry nulla
3. Mandi in errore la funzione `update()` della classe `CompositeCache` in modo che vada a stimolare il blocco catch sollevando l'eccezione `CacheException`

```
public void put( Object key, Object val, IElementAttributes attr )
    throws CacheException
{
    if ( key == null )
    {
        throw new InvalidArgumentException( "Key must not be null" );
    }
    else if ( val == null )
    {
        throw new InvalidArgumentException( "Value must not be null" );
    }

    // Create the element and update. This may throw an IOException which
    // should be wrapped by cache access.
    try
    {
        CacheElement ce = new CacheElement( this.cacheControl.getCacheName(), (Serializable) key,
                                             (Serializable) val );

        ce.setElementAttributes( attr );

        this.cacheControl.update( ce );
    }
    catch ( Exception e )
    {
        throw new CacheException( e );
    }
}
```

Per cercare di migliorare il coverage è stata quindi estesa la test suite, aggiungendo due casi di test per la classe `JUnitTest` che presentano rispettivamente un entry nulla ed un entry con chiave nulla. In questo modo, il coverage per il metodo `put()` diventa pari all'85%:

```
public void put( Object key, Object val, IElementAttributes attr )
    throws CacheException
{
    if ( key == null )
    {
        throw new InvalidArgumentException( "Key must not be null" );
    }
    else if ( val == null )
    {
        throw new InvalidArgumentException( "Value must not be null" );
    }

    // Create the element and update. This may throw an IOException which
    // should be wrapped by cache access.
    try
    {
        CacheElement ce = new CacheElement( this.cacheControl.getCacheName(), (Serializable) key,
                                             (Serializable) val );

        ce.setElementAttributes( attr );

        this.cacheControl.update( ce );
    }
    catch ( Exception e )
    {
        throw new CacheException( e );
    }
}
```

Per sollevare `CacheException`, utilizziamo un approccio white-box ispezionando il codice del metodo `update()` della classe `CompositeCache`. Il metodo in analisi invoca il metodo `update()` della classe `IMemoryCache`, fornendo in input sempre l'oggetto `CacheElement`. Dopo diversi tentativi, non si è trovato il modo di sollevare `IOException`, e di conseguenza nel metodo `put()` non si riesce a passare nel blocco catch per sollevare l'eccezione desiderata, non riuscendo a migliorare ulteriormente il coverage.

In conclusione, considerando anche il commento leggibile sul codice: *“l’eccezione dovrebbe essere coperta dall’accesso alla cache”*, si è ritenuto che la copertura dei casi di test non fosse ulteriormente migliorabile, in quanto per migliorare il coverage del metodo `put()` non basta ampliare la test suite, ma forse ci troviamo davanti ad un “unreachable code”.

3 BOOKKEEPER

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza ottimizzato per carichi di lavoro real-time. Le classi considerate per le attività di testing sono: `DigestManager` e `ReadCache`.

La scelta di queste classi è basata sulla chiarezza della documentazione, sulla quantità/qualità dei commenti, e sulla comprensibilità del ruolo e delle funzioni svolte dalla classe all'interno del progetto.

I motivi per cui è stato utilizzato questo approccio, anziché basarsi sulle metriche calcolate durante l'analisi del codice, sono principalmente due:

1. Nell'analisi del software effettuata, le metriche sono state calcolate soltanto per le release fino alla 4.5.0., questo perché è necessario utilizzare i ticket di Jira andando ad effettuare un mapping con i relativi commit su GitHub. Di conseguenza, abbiamo a disposizione su Jira solamente i ticket fino alla release 4.5.0 perché Bookkeeper dal 2017 è migrato su GitHub.
2. Si è utilizzato un approccio di tipo black-box durante il Domain Partitioning, per poi passare ad un approccio white-box nel miglioramento della test-suite a seguito dell'analisi del coverage e del mutation testing. Per questo motivo, sono state selezionate quelle classi con la maggiore chiarezza nelle specifiche, e con le quali si aveva una maggiore familiarità.

3.1 READCACHE

La classe `ReadCache` implementa una cache di lettura, ovvero uno spazio di memoria suddiviso in diversi segmenti, che permette un accesso più rapido alle entry. Ogni entry viene identificata univocamente dall'ID del ledger e dall'ID assegnato all'entry stessa all'interno di quel ledger, e prima di essere scritta in modo persistente sul log, viene inserita all'interno della cache.

Per questa classe si è analizzato il comportamento dei metodi `put()` e `get()`, costruendo dei casi di test che andassero a stimolare principalmente questi due metodi.

3.1.1 `public void put(long ledgerId, long entryId, ByteBuf entry)`

Questo metodo copia nella cache il contenuto del `ByteBuf entry`, identificato da `entryId` e da scrivere nel ledger specificato da `ledgerId`.

Applichiamo *Domain Partitioning* inizialmente basandosi sulla *semantica* del nome per individuare i valori che i parametri possono assumere. I parametri `ledgerId` ed `entryId` sono degli *identificativi*, per cui ci si aspetta che debbano necessariamente essere non negativi. Per quanto riguarda invece il parametro `entry` non possiamo far altro che considerare un'istanza valida, una non valida ed una nulla trattandosi di un oggetto complesso. La prima partizione in classi di equivalenza è data da:

- `ledgerId`: {<0, >=0}
- `entryId`: {<0, >=0}
- `entry`: {valid, invalid, null}
 - invalid: entry con dimensione superiore a quella della cache istanziata

Per lo sviluppo dei casi di test si è seguito un approccio di tipo *unidimensionale*, considerando quindi i vari parametri in maniera del tutto indipendente e cercando di coprire con almeno un caso di test tutte le classi di equivalenza definite. Per la scelta dei valori rappresentativi di ogni partizione individuata si applica *Boundary Values Analysis*, che ha portato allo sviluppo dei seguenti casi di test:

- {`ledgerId`, `entryId`, `entry`}
 - {0, -1, null_entry} → `NullPointerException`
 - {-1, 0, valid_entry} → `IllegalArgumentException`
 - {1, 0, valid_entry} → success
 - {1, 1, invalid_entry} → `IndexOutOfBoundsException`

Il comportamento atteso da parte dei casi di test è stato recuperato tramite un approccio white-box, ispezionando il codice e individuando le eccezioni sollevate in base ai valori utilizzati dalla test suite.

L'assunzione sugli indici è parzialmente verificata, infatti per `ledgerId` si controlla che il valore sia "≥ 0", mentre per `entryId` non c'è alcun vincolo sul valore che può assumere.

I test selezionati per il metodo `put()` permettono di raggiungere una statement coverage pari al 76.2% ed una branch coverage pari al 75% [Figura 2]. Analizzando il report sul coverage fornito da JaCoCo [Figura 3], si nota che le istruzioni a riga 122 e 123 non vengono testate, di conseguenza è necessario stressare ulteriormente la classe per coprire queste istruzioni che utilizzano i segmenti e gli indici della cache. Per aumentare la coverage, consideriamo l'inserimento consecutivo di molteplici entry nella cache.

Introduciamo un nuovo parametro `expectedNumberEntries`, che utilizziamo per attivare o meno il ciclo `for` che effettua il `put` delle entry in cache, e per specificare quante put portare a termine. Per i casi di test già sviluppati, `expectedNumberEntries` è stato impostato a 0, in modo da non alterare i risultati ottenuti tramite *domain partitioning*, andando ad inserire solamente una entry.

La test suite viene estesa aggiungendo un nuovo caso di test che presenta `expectedNumberEntries` impostato a 1, in modo da realizzare `N` put, e al termine si controlla che nella cache vi siano proprio `N` entry.

Dopo il miglioramento della test suite, il report fornito da JaCoCo mostra una statement coverage del 100% e branch coverage del 75% [Figura 4][Figura 4]. Analizzando il report sul coverage, si nota che, nonostante la test suite sia stata ampliata, non si è riuscito a coprire con i casi di test implementati il branch di riga 113:

```
o [offset+entrySize>segmentSize == false].
```

Si è cercato quindi di includere un nuovo caso di test in cui `offset + entrySize <= segmentSize`, ma anche analizzando nello specifico l'implementazione del metodo `put`, non si è riuscito a sviluppare un caso di test che percorra quel branch.

Il branch di riga 113 sembrerebbe essere un branch non percorribile, in quanto:

1. se abbiamo `offset + entrySize <= segmentSize` a riga 94 si entra nel ramo `else` di riga 96, al termine del quale c'è l'istruzione `return`
2. se abbiamo `offset + entrySize > segmentSize` a riga 94 si entra nel ramo `if`, l'esecuzione prosegue, ma non si è trovato il modo di avere `offset + entrySize > segmentSize` a riga 94 e successivamente `offset + entrySize <= segmentSize` a riga 113

Per migliorare l'adeguatezza dei casi di test, si va ad applicare *Mutation Testing* tramite il framework *Pit*. I risultati [

Figura 6] hanno mostrato come le mutazioni di riga 94 e 113 non siano state rilevate. Si è cercato quindi di migliorare l'implementazione del test program utilizzando il costruttore `ReadCache(allocator, maxCacheSize, maxSegmentSize)` che permette di specificare la massima dimensione assegnabile ai segmenti, anziché utilizzare quella di default come in precedenza. Sono stati inclusi quindi almeno un caso di test in cui `entrySize>maxSegmentSize>maxCacheSize`, uno in cui

`entrySize<maxSegmentSize<maxCacheSize` ed uno in cui `entrySize>maxSegmentSize<maxCacheSize`.

Tuttavia anche a seguito di tali modifiche sia la branch coverage che il mutation coverage sono rimasti invariati. Per migliorare i risultati ottenuti sarebbe quindi necessario uno studio approfondito del codice e del funzionamento riguardo il calcolo dell'offset e della dimensione del segmento. Tale analisi richiederebbe però un *effort* non banale, motivo per cui si è deciso di mantenere la test suite minimale, non includendo gli ultimi casi di test introdotti. In conclusione, la test suite è stata considerata come sufficientemente adeguata, avendo comunque raggiunto una statement coverage del 100% ed una branch coverage del 75%.

3.1.2 `public ByteBuf get(long ledgerId, long entryId)`

Questo metodo prende in input l'identificativo del ledger e l'identificativo dell'entry, restituendo il contenuto dell'entry se questa è presente in cache. Tramite *domain partitioning* otteniamo una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei parametri in input:

- `ledgerId` {<0,>=0}
- `entryId` {<0,>=0}

Per testare il comportamento del metodo `get`, in fase di configurazione andiamo ad inserire una entry con `(ledgerId,entryId)=(1,1)` nella cache tramite `put`, in modo da poter recuperare il suo contenuto. Per la realizzazione della test suite, introduciamo il parametro `expectedEntry`, in modo da verificare il risultato restituito dalla `get` includendo un caso di test in cui la `get` ha successo, ovvero l'entry cercata viene trovata in cache, ed un caso di test in cui la `get` fallisce, ovvero l'entry cercata non viene trovata nella cache. Applicando *boundary values* ed utilizzando un criterio di selezione *unidimensionale*, sono stati quindi definiti i seguenti test case:

- **{`ledgerId`,`entryId`,`expectedEntry`}**
 - {-1,0,NULL_ENTRY} → `IllegalArgumentException`
 - {0,-1,NULL_ENTRY} → `entry not found`
 - {1,0,NULL_ENTRY} → `entry not found`
 - {1,1,ENTRY} → `success`

Come già detto per il metodo `put`, *bookkeeper* ammette valori negativi come identificativi per le entry, almeno per quanto riguarda la classe `ReadCache`, di conseguenza il secondo caso di test non solleverà un `IllegalArgumentException`, ma semplicemente la entry ricercata non è presente in cache e quindi il metodo ritornerà una entry nulla.

Analizzando il report sul coverage, si vede come la test suite sia sufficientemente adeguata, questo perché è stato raggiunto il 100% sia per lo *statement coverage* che per il *branch coverage*. [Figura 7][Figura 8].

Per un'ultima analisi è stato eseguito il *mutation testing*. Il report di *Pit* [Figura 9] ha mostrato come non tutti i mutanti vengano rilevati, ad esempio come nella mutazione di riga 151 in cui è stata rimossa la chiamata `lock.readLock().unlock()`. Quello che possiamo dire è che questo è un comportamento atteso in quanto anche senza togliere il lock di lettura non abbiamo errori e e quindi non c'è nessun comportamento diverso rispetto a come è stato esplorato e testato il sistema.

Il mutante di riga 138 porta alla sostituzione della sottrazione con l'addizione, per provare ad ucciderlo si è pensato di ampliare la test suite per stimolare maggiormente il controllo sui segmenti. Come già fatto per il metodo `put`, aggiungiamo un ulteriore caso di test in cui effettuiamo molteplici `get` consecutive di entry differenti.

Nonostante vari tentativi, il *mutation coverage* è rimasto invariato, di conseguenza, avendo già raggiunto una coverage del 100%, si ritiene sufficiente la test suite minimale che è stata implementata.

3.2 DIGESTMANAGER

Bookkeeper supporta e utilizza diversi tipi di digest per il controllo di integrità delle entry che vengono memorizzate nei server (bookie). La classe `DigestManager` viene utilizzata:

1. Lato client per calcolare il digest da allegare al pacchetto da inviare al server
2. Lato server per verificare la correttezza del pacchetto ricevuto, controllando il digest ed estraendo l'entry originale

3.2.1 `public ByteBufList computeDigestAndPackageForSending(long entryId, long lastAddConfirmed, long length, ByteBuf data)`

Questo metodo permette al client di preparare l'entry da inviare al server, andando a calcolare il digest e concatenarlo con il buffer di dati associato all'entry. Tramite *domain partitioning* otteniamo una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei parametri in input:

- `ledgerId` {<0,>=0}
- `entryId` {<=lastAddConfirmed,>lastAddConfirmed}

- `length {=0,>0}`
- `data {null,valid,invalid}`
 - `buffer` che rappresenta l'entry a cui verrà concatenato il digest
 - `valid: data.len > 0`
 - `invalid: data.len = 0`

Per lo sviluppo dei casi di test si è seguito un approccio di tipo unidimensionale, mentre per la scelta dei valori rappresentativi di ogni partizione individuata si applica Boundary Values Analysis, arrivando allo sviluppo dei seguenti casi di test:

- **`{lastAddConfirmed,entryId,length,data}`**
 - `{-1,-1,0,null}` → `IllegalArgumentException`
 - `{0,1,0,data.len=0}` → `success`
 - `{1,2,1,data.len>0}` → `success`

Il comportamento atteso da parte dei casi di test è stato recuperato tramite un approccio white-box, ispezionando il codice e individuando le eccezioni sollevate in base ai valori utilizzati dalla test suite. I test selezionati per il metodo `computeDigestAndPackageForSending()` permettono di raggiungere una statement coverage del 70% e una branch coverage del 37% [Figura 10].

Analizzando il report [Figura 11] si può notare la presenza del parametro `useV2Protocol`, inizializzato nel costruttore della classe ed indirettamente collegato al metodo in analisi. Possiamo quindi raffinare la nostra test suite istanziando in fase di configurazione la classe `DigestManager` non più con `useV2Protocol` fisso a `false`, ma utilizzando un nuovo parametro variabile aggiunto a quelli utilizzati dai vari casi di test:

- **`{lastAddConfirmed,entryId,length,data,useV2Protocol}`**
 - `{-1,-1,0,null,false}` → `IllegalArgumentException`
 - `{0,1,0,data.len=0,true}` → `success`
 - `{1,2,1,data.len>0,false}` → `success`

Possiamo migliorare ulteriormente il livello del coverage ampliando la nostra test suite, utilizzando come entry rispettivamente un `WrappedBuffer` e un `CompositeByteBuf`, così da attraversare i branch a riga 118 e 123:

- **`{lastAddConfirmed,entryId,length,data,useV2Protocol}`**
 - `{-1,-1,0,null,false}` → `IllegalArgumentException`
 - `{0,1,0,data.len=0,true}` → `success`
 - `{1,2,1,data.len>0,false}` → `success`
 - `{0,1,0,WrappedEntry,true}` → `success`
 - `{0,1,0,CompositeByteBufEntry,true}` → `success`

A seguito di questi miglioramenti introdotti, riusciamo a raggiungere il 96% di statement coverage e l'87% di branch coverage [Figura 12]. Si è provato a raggiungere il 100% di coverage senza successo, non riuscendo a coprire un solo branch dei 4 presenti nell'istruzione composta a riga 118.

Per effettuare un'ultima valutazione sull'adeguatezza della test suite è stato applicato il *mutation testing*. Il report di *Pit* [Figura 13] mostra che non sono state rilevate le mutazioni di tipo *remove call* (riga 115,124,126,128) e quelle su `useV2Protocol` (riga 105). Riguardo quest'ultimo la mutazione introdotta è una *negated conditional* che va ad istanziare un buffer. La mutazione non viene rilevata perché cambia solo la classe utilizzata per istanziare il buffer, ma la dimensione e i dati scritti al di sopra non cambiano. Per le mutazioni di tipo *remove call*, non abbiamo modo per ucciderle, questo perché non sono presenti i dovuti controlli sul codice, ma il comportamento atteso è lo stesso con/senza quelle precise invocazioni.

In conclusione, la test suite dovrebbe risultare comunque adeguata avendo raggiunto quasi il 100% di coverage per entrambe le metriche utilizzate.

3.2.2 `private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)`

Questo metodo permette al server, in ricezione, di verificare il digest del pacchetto ricevuto dal client ed ottenere l'entry corrispondente, rimuovendo l'header dovuto al digest.

Essendo un metodo privato, non può essere invocato direttamente, di conseguenza andiamo a testarlo sfruttando il metodo pubblico `verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)`, che verifica la correttezza del digest e restituisce il buffer di dati associato all'entry.

Tramite *domain partitioning* otteniamo una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei parametri in input:

- `entryId` {<0,>=0}
- `dataReceived` {null,valid,invalid}
 - buffer che rappresenta l'entry ricevuta, compreso il digest che è stato concatenato

Tramite un approccio white-box, si nota che la correttezza dell'entry dipende da `ledgerId` e `entryId` mantenuti, che devono matchare quelli della classe stessa, quindi:

- valid: (`dataReceived.ledgerId` = `ledgerId` AND `dataReceived.entryId` = `entryId`)
- invalid: (`dataReceived.ledgerId` != `ledgerId` AND `dataReceived.entryId` != `entryId`) OR (`dataReceived.ledgerId` = `ledgerId` AND `dataReceived.entryId` != `entryId`) OR (`dataReceived.ledgerId` != `ledgerId` && `dataReceived.entryId` = `entryId`)

Andiamo quindi ad includere il parametro `ledgerId`, istanziato nel costruttore e indirettamente collegato al metodo in analisi. Individuiamo anche il parametro `skipEntryIdCheck`, ma non lo inseriamo nei parametri perché il metodo pubblico invoca quello privato in analisi sempre con `skipEntryIdCheck=false`.

Per lo sviluppo dei casi di test si è seguito un approccio di tipo *unidimensionale*, mentre per la scelta dei valori rappresentativi di ogni partizione individuata si applica *Boundary Values Analysis*, arrivando allo sviluppo dei seguenti casi di test:

- **{entryId,dataReceived,ledgerId}**
 - {-1,null,-1}
 - `NullPointerException`
 - {0,`DataRec.ledgerId!=ledgerId && DataRec.entryId=entryId,0`}
 - success
 - {1,`DataRec.ledgerId!=ledgerId && DataRec.entryId!=entryId,1`}
 - `BKDigestMatchException`
 - {0,`DataRec.ledgerId=ledgerId && DataRec.entryId!=entryId,0`}
 - `BKDigestMatchException`
 - {1,`DataRec.ledgerId!=ledgerId && DataRec.entryId=entryId,1`}
 - `BKDigestMatchException`

Come per gli altri metodi testati, anche qui il comportamento atteso da parte dei vari casi di test è stato recuperato tramite un approccio white-box, ispezionando il codice e individuando le eccezioni sollevate in base ai valori utilizzati dalla test suite.

Il report di Jacoco [Figura 14] ha mostrato il raggiungimento di una statement coverage del 76% e branch coverage del 70%. Analizzando il report [Figura 15] si può osservare come i casi di test implementati non vadano a coprire i branch di riga 165,181 e 198.

Riguardo riga 165 e 181, è possibile aumentare il coverage ampliando la test suite realizzata, in particolare è necessario avere due test per cui:

1. Una entry ha dimensione minore del previsto, in modo da verificare la condizione `(METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()`
2. Una entry ha digest differente da quella registrata, in modo da verificare la condizione `digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0`

Per soddisfare il secondo caso, andiamo a modificare il metodo `generateEntryWithDigest()` aggiungendo la selezione del `DigestType`, non introdotto in precedenza perché ritenuto non necessario, creando perciò tutte entry con lo stesso tipo di digest.

La test suite migliorata per aumentare il coverage sarà composta dai seguenti casi di test:

- **{entryId,dataReceived,ledgerId}**
 - {-1,null,-1}
→ NullPointerException
 - {0,DataRec.ledgerId!=ledgerId && DataRec.entryId=entryId,0}
→ success
 - {1,DataRec.ledgerId!=ledgerId && DataRec.entryId!=entryId,1}
→ BKDigestMatchException
 - {0,DataRec.ledgerId=ledgerId && DataRec.entryId!=entryId,0}
→ BKDigestMatchException
 - {1,DataRec.ledgerId!=ledgerId && DataRec.entryId=entryId,1}
→ BKDigestMatchException
 - {1,DataRec.readableBytes()<METADATA_LENGTH+macCodeLength,1}
→ BKDigestMatchException
 - {1,DataRec.readableBytes()>METADATA_LENGTH+macCodeLength && digestManager.digestType!=entry.digestType,1}
→ BKDigestMatchException

A seguito di questi miglioramenti introdotti, riusciamo a raggiungere il 100% di statement coverage e il 90% di branch coverage [Figura 16]. Analizzando il report di JaCoCo [Figura 17] possiamo notare come grazie all'aggiunta di due nuovi casi di test, i branch di riga 165 e 181 vengono coperti, e di conseguenza vengono percorse anche le istruzioni a riga 166-167 e 182-183.

Non è però possibile raggiungere il 100% di coverage perché non siamo in grado di coprire l'unico branch su 4 non esplorato di riga 198, questo perché come già detto il parametro `skipEntryIdCheck` viene utilizzato sempre a `false` dal metodo pubblico che utilizziamo per testare il metodo in analisi.

questo test genera un entry con digest diverso da quello usato dalla classe `DigestManager`, di conseguenza la condizione è verificata e viene sollevata una `BKDigestMatchException`

Per valutare e migliorare ulteriormente l'adeguatezza della test suite è stato applicato *mutation testing*.

Il report di *Pit* [Figura 18] ha mostrato come diversi mutanti non siano stati rilevati.

Un mutante non-killed interessante mostrato dal report si trova a riga 181, mutante di tipo *negated conditional*, per cui non viene rilevato un diverso comportamento cambiando la boundary dell'istruzione `if(digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0)`.

Per rilevare e uccidere la mutazione è necessario aggiungere un caso di test per cui l'esito del SUT deve essere differente da quello del SUT mutato. Includiamo il seguente caso di test:

- {1,DataRec.ledgerId=ledgerId && DataRec.entryId=entryId && DataRec.DigestType!=DigestType,1}
→ BKDigestMatchException

Il caso di test aggiunto genera un entry con digest differente da quello usato dalla classe `DigestManager`, di conseguenza la condizione dell'if è verificata e verrà sollevata una `BKDigestMatchException`.

Il framework PIT altera la condizione dell'if da "!=" a "==", così facendo il flusso d'esecuzione del SUT mutato non solleverà una `BKDigestException`, ma l'esecuzione andrà a buon fine.

Per rimuovere questo mutante, `ledgerId` e `entryId` dell'entry sono diversi da quelli della classe `DigestManager`, in modo tale che il flusso d'esecuzione del SUT mutato non entri nelle due if di riga 192 e 198, che sono mutate anch'esse, portando a buon fine l'esecuzione senza sollevare alcuna eccezione.

Il nuovo report Pit [Figura 19] mostra come effettivamente la mutazione venga rilevata, mentre non si ottiene alcun miglioramento sul coverage analizzando il report JaCoCo, rimanendo sempre su una statement coverage del 100% e una branch coverage del 90%.

4 AVRO

Apache Avro è un sistema Open Source per la serializzazione dei dati basato su *schemi*, che vengono utilizzati per definire in che modo il flusso di dati deve essere scritto, o il modo in cui deve essere letto. Durante l'esecuzione delle due operazioni, lo *schema* è sempre presente.

Le classi considerate per le attività di testing sono: `BinaryData` e `RealmUtils`. La scelta di queste classi non si è basata sulle metriche ricavate durante l'analisi del codice, bensì sulla chiarezza della documentazione e del ruolo svolto all'interno del progetto. Il motivo principale, come già detto per Bookkeeper, per cui è stato utilizzato questo criterio di selezione, è dovuto al fatto che si è seguito un approccio di tipo black-box durante la prima fase di domain partitioning e boundary analysis, per poi migrare su un approccio più di tipo white-box nel miglioramento della test suite a seguito dei report Jacoco e Pit. Per questo le classi scelte sono state quelle con la maggiore chiarezza delle specifiche, e con le quali si aveva una maggiore conoscenza/familiarità.

4.1 BINARYDATA

Questa classe fornisce gli strumenti per la manipolazione di dati binari.

4.1.1 `Int compare(Decoders d, Schema schema)`

Per testare il metodo `compare` della classe `BinaryData`, essendo un metodo privato, utilizzeremo il metodo pubblico `compare(byte[] b1, int s1, byte[] b2, int s2, Schema schema)`, che confronta due array di byte sfruttando per il confronto lo schema fornito in input.

Tramite *domain partitioning* otteniamo una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei parametri in input:

- `b1 {null, b1.Type=schema.Type, b1.Type!=schema.Type}`
 - array di byte generato secondo lo schema `b2.Type`
- `s1 {<=0, >0}`
 - offset all'interno di `b1` da cui iniziare il confronto tra i due array
- `b2 {null, b2.Type=b1.Type=schema.Type, b2.Type=b1.Type!=schema.Type, b2.Type!=b1.Type}`
 - array di byte generato secondo lo schema `b2.Type`
- `s2 {<=s1, >s1}`
 - offset all'interno di `b2` da cui iniziare il confronto tra i due array
- `schema {ARRAY, BOOLEAN, BYTES, DOUBLE, ENUM, FIXED, FLOAT, INT, LONG, MAP, NULL, RECORD, STRING, UNION, other}`
 - schema utilizzato per confrontare i due array

Il partizionamento relativo al parametro `b2` è stato effettuato in base a `b1.Type` e `schema.Type`, assumendo quindi che il confronto non vada a buon fine se i due array non matchano lo schema di riferimento. Tramite un approccio white-box, però, si evince dal codice che il confronto non dipende dal parametro `Schema`, ma per due array uguali il confronto va a buon fine a prescindere dallo schema di riferimento utilizzato. Di conseguenza, l'assunzione fatta ha portato alla realizzazione di una test suite non proprio minimale, questo perché non si è tenuto conto del codice in fase di *domain partitioning*.

Per l'esecuzione del test sono stati aggiunti i metodi:

- `getS(Schema.Type type) :` crea un oggetto *schema* per lo specifico *Type* fornito in input. La creazione dell'oggetto avviene in due modi differenti:
 - Per i tipi di dato semplici, viene utilizzato il metodo `schema.create()`
 - Per i tipi di dato complesso, viene utilizzato il metodo `parse()` che, data una stringa in input, genera un determinato schema.
- `getBS(Schema.Type type, boolean createB1) :` crea un array di byte generato secondo lo schema fornito in input. Si utilizza il parametro booleano per poter creare array con valori differenti.

È stata aggiunta anche la classe `BinaryDataUtils`, che offre dei metodi di supporto per la creazione degli array di byte relativi ad un determinato tipo di schema.

Per lo sviluppo dei casi di test si è seguito un approccio di tipo unidimensionale, mentre per la scelta dei valori rappresentativi di ogni partizione individuata si applica Boundary Values Analysis, arrivando alla seguente test suite minimale:

- **{b1,b2,s1,s2,schema}**
 - null,null,-1,-2,RECORD → NullPointerException
 - b1.Type=FIXED,b2.Type=FIXED,0,0,FIXED → 0
 - b1.Type=ARRAY,b2.Type=ARRAY,1,2,INT → -1
 - b1.Type=INT,b2.Type=DOUBLE,0,0,ARRAY → AvroRuntimeException
 - b1.Type=MAP,b2.Type=DOUBLE,0,0,BOOLEAN → 0
 - b1.Type=NULL,b2.Type=DOUBLE,0,0,BYTES → NullPointerException
 - b1.Type=DOUBLE,b2.Type=DOUBLE,0,0,DOUBLE → -1
 - b1.Type=ENUM,b2.Type=ENUM,0,0,ENUM → 0
 - b1.Type=NULL,b2.Type=DOUBLE,0,0,FLOAT → NullPointerException
 - b1.Type=LONG,b2.Type=LONG,0,0,LONG → 1
 - b1.Type=INT,b2.Type=DOUBLE,0,0,MAP → AvroRuntimeException
 - b1.Type=INT,b2.Type=DOUBLE,0,0,NULL → 0
 - b1.Type=STRING,b2.Type=STRING,0,0,STRING → 35
 - b1.Type=INT,b2.Type=DOUBLE,0,0,UNION → AvroRuntimeException
 - b1.Type=INT,b2.Type=DOUBLE,0,0,other → -1

Come per gli altri metodi testati, anche qui il comportamento atteso da parte dei vari casi di test è stato recuperato tramite un approccio white-box, ispezionando il codice e individuando le eccezioni sollevate in base ai valori utilizzati dalla test suite.

Il report di JaCoCo [Figura 20] mostra il raggiungimento del 63% di statement coverage e del 48% di branch coverage. Analizzando il report [Figura 21], possiamo notare come molteplici istruzioni/branch non vengono coperte dalla test suite implementata, in particolare nei blocchi case relativi allo schema di riferimento di tipo *RECORD*,*ARRAY* e *FLOAT*.

Per aumentare la coverage, andiamo ad ampliare la nostra test suite, aggiungendo i seguenti casi di test:

- b1.Type=RECORD,b2.Type=RECORD,0,0,RECORD → 0
- b1.Type=ARRAY,b2.Type=ARRAY,0,0,ARRAY → 1
- b1.Type=FLOAT,b2.Type=FLOAT,0,0,FLOAT → 0

Il primo caso di test ci permette esplorare righe di codice all'interno del blocco *RECORD*, questo perché l'unico test con `schema.Type=Record` fallisce prima di invocare il metodo `compare()`, dovuto al fatto che i due array di byte sono entrambi nulli.

Il secondo caso di test utilizza per la comparazione due array di byte generati in maniera coerente con lo schema di riferimento, ma con offset differente, riuscendo ad esplorare alcune istruzioni/branch che con la suite minimale non venivano coperte.

Il terzo caso di test, analogamente al primo, viene introdotto per esplorare le righe di codice presenti all'interno del blocco *FLOAT*, non coperte in precedenza perché l'unico test presente era caratterizzato da un array di byte generato secondo lo schema *NULL*.

Il miglioramento della suite minimale ci porta a raggiungere 94% di statement coverage e il 87% di branch coverage [Figura 22]. Analizzando il report JaCoCo [Figura 23], possiamo vedere come molte istruzioni e branch vengono coperti grazie dall'aggiunta dei nuovi casi di test.

Non si è però riuscito a migliorare ulteriormente il coverage per i seguenti motivi:

- Nel blocco "*RECORD*", non è possibile entrare nell'if a riga 90 perché non si è trovato il modo di manipolare l'ordinamento dei campi all'interno dello schema di riferimento
- Nel blocco "*ARRAY*", non è possibile coprire completamente i branch di riga 118-126, questo perché non si conosce nel dettaglio il comportamento dei `Decoders` utilizzati dalla funzione per leggere i byte

Nonostante numerosi tentativi, non si è riuscito a migliorare ulteriormente il coverage, questo perché non si conosce nel dettaglio l'implementazione e il comportamento dei `Decoders`, non riuscendo di conseguenza a coprire totalmente i vari branch del metodo.

Per valutare e migliorare ulteriormente l'adeguatezza della test suite è stato applicato *mutation testing*. Il report di *Pit* [Figura 24] ha mostrato come diversi mutanti non siano stati rilevati. Tralasciando i mutanti che vengono generati dove non siamo riusciti a garantire la copertura tramite la test suite, introduciamo dei nuovi casi di test per killare i mutanti di tipo *replace int return* e *negated conditional*:

```
➤ b1.Type=BOOLEAN,b2.Type=BOOLEAN,0,0,BOOLEAN → 1
➤ b1.Type=ARRAY,b2.Type=ARRAY,0,1,ARRAY → 1
➤ b1.Type=FLOAT,b2.Type=FLOAT,0,0,FLOAT → -1
➤ b1.Type=UNION,b2.Type=UNION,1,0,UNION → 1
➤ b1.Type=FIXED,b2.Type=FIXED,0,0,FIXED → -1
➤ b1.Type=STRING,b2.Type=STRING,1,0,STRING → -35
➤ b1.Type=ARRAY,b2.Type=ARRAY,1,0,ARRAY → 0
➤ b1.Type=ARRAY,b2.Type=ARRAY,0,0,ARRAY → -1
➤ b1.Type=LONG,b2.Type=LONG,1,0,LONG → 0
```

La maggior parte dei test introdotti presentano i due array di byte usati per la comparazione con un offset differente, oppure vengono costruiti proprio con valori diversi, in modo da poter killare i mutanti elencati. Passiamo da 86 a 107 mutanti killed per la classe in analisi [Figura 25], riuscendo anche ad incrementare la statement coverage da 94% a 96%.

Dalle considerazioni già fatte in precedenza, sia per la complessità del metodo in analisi, che per le elevate conoscenze implementative richieste, si ritiene sufficientemente adeguato il risultato ottenuto, questo perché nonostante le molte difficoltà incontrate, si è arrivati comunque ad avere una copertura quasi totale, almeno a livello degli statement.

4.2 SPECIFICDATA

Questa classe offre delle utility per la generazione e la gestione di oggetti di tipo *schema* per tipi di dato specifici del linguaggio Java.

4.2.1 protected createSchema(final String realmPath, final String groupKey)

Per testare il metodo `createSchema` della classe `SpecificData`, essendo un metodo protetto, andiamo ad utilizzare il metodo pubblico `getSchema(java.lang.reflect.Type type)`, che ci restituisce lo schema associato alla tipologia di classe Java fornita in input.

Tramite *domain partitioning* otteniamo una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei parametri in input:

- `type {null,Array,Boolean,Bytes,Double,Enum,Fixed,Float,Int,Long,Map,Void,Record,String,Union,Other}`

Per l'esecuzione del test è stato aggiunto il metodo:

- `getClassType(String classTypeString)` : restituisce il tipo di dato Java associato alla stringa fornita in input, necessario come parametro per il metodo `getSchema()`.
 - Per i tipi di dato semplice viene restituito, come tipo di dato, la classe Java dell'oggetto (`object.class`)
 - Per i tipi di dato complessi, map e collection, viene utilizzata la reflection su degli attributi privati della classe di test per ottenere la tipologia delle due variabili a runtime

Per lo sviluppo dei casi di test si è seguito un approccio di tipo *unidimensionale*, mentre per la scelta dei valori rappresentativi di ogni partizione individuata si applica *Boundary Values Analysis*, arrivando alla seguente test suite minimale:

- **{type}**
 - `getClassType("null")` → `AvroTypeException`
 - `getClassType("array")` → `Schema.Type.ARRAY`
 - `getClassType("boolean")` → `Schema.Type.BOOLEAN`
 - `getClassType("bytes")` → `Schema.Type.BYTES`
 - `getClassType("double")` → `Schema.Type.DOUBLE`

➤ getClassType("enum")	→ AvroRuntimeException
➤ getClassType("fixed")	→ AvroRuntimeException
➤ getClassType("float")	→ Schema.Type.FLOAT
➤ getClassType("int")	→ Schema.Type.INT
➤ getClassType("long")	→ Schema.Type.LONG
➤ getClassType("map")	→ Schema.Type.MAP
➤ getClassType("void")	→ Schema.Type.NULL
➤ getClassType("record")	→ AvroRuntimeException
➤ getClassType("string")	→ Schema.Type.STRING
➤ getClassType("union")	→ AvroRuntimeException
➤ getClassType("other")	→ AvroRuntimeException

Come per gli altri metodi testati, anche qui il comportamento atteso da parte dei vari casi di test è stato recuperato tramite un approccio white-box, ispezionando il codice e individuando le eccezioni sollevate in base ai valori utilizzati dalla test suite.

Il report di JaCoCo [\[Figura 26\]](#) mostra il raggiungimento del 73% di statement coverage e del 72% di branch coverage. Analizzando il report [\[Figura 27\]](#), possiamo notare come non vengono coperti completamente tutti i branch da riga 374 a 384, e seguendo un approccio white-box, possiamo aumentare la coverage ampliando la test suite, in modo tale da esplorare ogni condizione dei branch in esame:

➤ Integer.TYPE	→ Schema.Type.INT
➤ Long.TYPE	→ Schema.Type.LONG
➤ Float.TYPE	→ Schema.Type.FLOAT
➤ Double.TYPE	→ Schema.Type.DOUBLE
➤ Boolean.TYPE	→ Schema.Type.BOOLEAN
➤ Void.TYPE	→ Schema.Type.NULL
➤ getClassType("mapInt")	→ AvroTypeException

L'ultimo caso di test un'array di interi anziché di stringhe, così da riuscire a sollevare l'eccezione a riga 378. Il miglioramento della suite minimale ci porta a raggiungere il 79% di statement coverage e l'87% di branch coverage [\[Figura 28\]](#).

Analizzando il report di JaCoCo [\[Figura 29\]](#), vediamo come la nostra test suite non offre alcuna copertura degli statement presenti nel blocco *try* di riga 408. Nonostante numerosi tentativi, qualsiasi classe sia stata scelta, se non c'è uno schema predefinito per essa, la chiamata a riga 408 solleva *AvroRuntimeException* in ogni caso, rendendo anche il secondo blocco *catch* di riga 417 non raggiungibile, e di conseguenza ci risulta difficile, se non impossibile, migliorare il coverage.

Per valutare e migliorare ulteriormente l'adeguatezza della test suite applichiamo *mutation testing*.

Il report di *Pit* [\[Figura 30\]](#) ci dice che la nostra test suite è sufficientemente adeguata riuscendo a killare tutti i mutanti rilevati, ad eccezione ovviamente di quelli a riga 401-411-421 dove non abbiamo copertura.

5 LINK

5.1 JCS*

- Repository GitHub: <https://github.com/jacopofabi/JCS-ISW2>
- Travis CI: <https://travis-ci.com/github/jacopofabi/JCS-ISW2>

5.2 BOOKKEEPER

- Repository GitHub: <https://github.com/jacopofabi/bookkeeper>
- Travis CI: <https://travis-ci.com/github/jacopofabi/bookkeeper>
- Sonarcloud: https://sonarcloud.io/dashboard?id=jacopofabi_bookkeeper

5.3 AVRO

- Repository GitHub: <https://github.com/jacopofabi/avro>
- Travis CI: <https://travis-ci.com/github/jacopofabi/avro>
- Sonarcloud: https://sonarcloud.io/dashboard?id=jacopofabi_avro

**Per l'attività di JCS non troviamo su SonarCloud il report del coverage perché, non avendo a disposizione il codice sorgente del progetto, il report è stato generato in locale dopo ogni modifica sfruttando il "jar" instrumentato per l'analisi dell'andamento sul coverage.*

6 APPENDICE

Figura 1 – Coverage JCS

	Coverage	Covered Instructions	Missed Instructions	Total Instructions
CacheAccess	23,2 %	76	251	327
CacheAccess(CompositeCache)	100,0 %	6	0	6
dispose()	100,0 %	4	0	4
get(Object)	100,0 %	13	0	13
getStats()	100,0 %	4	0	4
put(Object, Object)	100,0 %	8	0	8
remove(Object)	100,0 %	7	0	7
<clinit>()	90,9 %	10	1	11
put(Object, Object, IElementAttributes)	60,0 %	24	16	40
defineRegion(String)	0,0 %	0	8	8
defineRegion(String, ICompositeCacheAttributes)	0,0 %	0	9	9
defineRegion(String, ICompositeCacheAttributes, IElementAttributes)	0,0 %	0	10	10
ensureCacheManager()	0,0 %	0	19	19
getAccess(String)	0,0 %	0	8	8
getAccess(String, ICompositeCacheAttributes)	0,0 %	0	9	9
clear()	0,0 %	0	11	11
destroy()	0,0 %	0	11	11
destroy(Object)	0,0 %	0	7	7
freeMemoryElements(int)	0,0 %	0	31	31
getCacheAttributes()	0,0 %	0	4	4
getCacheElement(Object)	0,0 %	0	6	6
getDefaultElementAttributes()	0,0 %	0	4	4
getElementAttributes()	0,0 %	0	4	4
getElementAttributes(Object)	0,0 %	0	16	16
getStatistics()	0,0 %	0	4	4
putSafe(Object, Object)	0,0 %	0	25	25
remove()	0,0 %	0	3	3
resetElementAttributes(Object, IElementAttributes)	0,0 %	0	30	30
resetElementAttributes(IElementAttributes)	0,0 %	0	5	5
setCacheAttributes(ICompositeCacheAttributes)	0,0 %	0	5	5
setDefaultElementAttributes(IElementAttributes)	0,0 %	0	5	5

Figura 2 – Coverage put

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
size()		0%		0%	4	9	1
put(long, long, ByteBuf)		76%		75%	1	3	0
close()		100%		n/a	0	0	0
ReadCache(ByteBufAllocator, long)		100%		n/a	0	0	0
count()		100%		100%	0	0	0
get(long, long)		100%		100%	0	0	0
ReadCache(ByteBufAllocator, long, int)		100%		100%	0	0	0
Total	77 of 356	78%	7 of 18	61%	5	12	1

Figura 3 – Analisi coverage del metodo put

```

86. public void put(long ledgerId, long entryId, ByteBuf entry) {
87.     int entrySize = entry.readableBytes();
88.     int alignedSize = align64(entrySize);
89.
90.     lock.readLock().lock();
91.
92.     try {
93.         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94.         if (offset + entrySize > segmentSize) {
95.             // Roll-over the segment (outside the read-lock)
96.         } else {
97.             // Copy entry into read cache segment
98.             cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99.                 entry.readableBytes());
100.             cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101.             return;
102.         }
103.     } finally {
104.         lock.readLock().unlock();
105.     }
106.
107.     // We could not insert in segment, we to get the write lock and roll-over to
108.     // next segment
109.     lock.writeLock().lock();
110.
111.     try {
112.         int offset = currentSegmentOffset.getAndAdd(entrySize);
113.         if (offset + entrySize > segmentSize) {
114.             // Rollover to next segment
115.             currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116.             currentSegmentOffset.set(alignedSize);
117.             cacheIndexes.get(currentSegmentIdx).clear();
118.             offset = 0;
119.         }
120.
121.         // Copy entry into read cache segment
122.         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(), entry.readableBytes());
123.         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
124.     } finally {
125.         lock.writeLock().unlock();
126.     }
127. }

```

Figura 4 – Report coverage put (dopo il miglioramento della test suite)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Ctxty	Missed Lines	Missed Methods
put(long, long, ByteBuf)	<div><div></div></div>	100%	<div><div></div></div>	75%	1 3	0 21	0 1
ReadCache(ByteBufAllocator, long, int)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 12	0 1
get(long, long)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 3	0 13	0 1
count()	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 6	0 1
ReadCache(ByteBufAllocator, long)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
close()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
size()	<div><div></div></div>	0%	<div><div></div></div>	0%	4 4	9 9	1 1
Total	48 of 356	86%	7 of 18	61%	5 16	9 65	1 7

Figura 5 - Analisi coverage del metodo put (dopo miglioramento della test suite)

```

86. public void put(long ledgerId, long entryId, ByteBuf entry) {
87.     int entrySize = entry.readableBytes();
88.     int alignedSize = align64(entrySize);
89.
90.     lock.readLock().lock();
91.
92.     try {
93.         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94.         if (offset + entrySize > segmentSize) {
95.             // Roll-over the segment (outside the read-lock)
96.         } else {
97.             // Copy entry into read cache segment
98.             cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99.                 entry.readableBytes());
100.             cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101.             return;
102.         }
103.     } finally {
104.         lock.readLock().unlock();
105.     }
106.
107.     // We could not insert in segment, we to get the write lock and roll-over to
108.     // next segment
109.     lock.writeLock().lock();
110.
111.     try {
112.         int offset = currentSegmentOffset.getAndAdd(entrySize);
113.         if (offset + entrySize > segmentSize) {
114.             // Rollover to next segment
115.             currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116.             currentSegmentOffset.set(alignedSize);
117.             cacheIndexes.get(currentSegmentIdx).clear();
118.             offset = 0;
119.         }
120.
121.         // Copy entry into read cache segment
122.         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(), entry.readableBytes());
123.         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
124.     } finally {
125.         lock.writeLock().unlock();
126.     }
127. }

```

Figura 6 – Mutation testing put

```

86 public void put(long ledgerId, long entryId, ByteBuf entry) {
87     int entrySize = entry.readableBytes();
88     int alignedSize = align64(entrySize);
89
90 1 lock.readLock().lock();
91
92     try {
93         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94 3 if (offset + entrySize > segmentSize) {
95         // Roll-over the segment (outside the read-lock)
96     } else {
97         // Copy entry into read cache segment
98         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99             entry.readableBytes());
100         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101         return;
102     }
103 } finally {
104 1 lock.readLock().unlock();
105 }
106
107 // We could not insert in segment, we to get the write lock and roll-over to
108 // next segment
109 1 lock.writeLock().lock();
110
111     try {
112         int offset = currentSegmentOffset.getAndAdd(entrySize);
113 3 if (offset + entrySize > segmentSize) {
114         // Rollover to next segment
115 2 currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116 1 currentSegmentOffset.set(alignedSize);
117 1 cacheIndexes.get(currentSegmentIdx).clear();
118         offset = 0;
119     }

```

Figura 7 - Coverage get













Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
size()		0%		0%	4	4	9	9	1	1
close()		100%		n/a	0	1	0	2	0	1
ReadCache(ByteBufAllocator,long)		100%		n/a	0	1	0	2	0	1
count()		100%		100%	0	2	0	6	0	1
get(long,long)		100%		100%	0	3	0	13	0	1
ReadCache(ByteBufAllocator,long,int)		100%		100%	0	2	0	12	0	1
put(long,long,ByteBuf)		100%		75%	1	3	0	21	0	1
Total	48 of 356	86%	7 of 18	61%	5	16	9	65	1	7

Figura 8 – Analisi coverage get

```

129.     public ByteBuf get(long ledgerId, long entryId) {
130.         lock.readLock().lock();
131.
132.         try {
133.             // We need to check all the segments, starting from the current one and looking
134.             // backward to minimize the
135.             // checks for recently inserted entries
136.             int size = cacheSegments.size();
137.             for (int i = 0; i < size; i++) {
138.                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
139.
140.                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
141.                 if (res != null) {
142.                     int entryOffset = (int) res.first;
143.                     int entryLen = (int) res.second;
144.
145.                     ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
146.                     entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
147.                     return entry;
148.                 }
149.             }
150.         } finally {
151.             lock.readLock().unlock();
152.         }
153.
154.         // Entry not found in any segment
155.         return null;
156.     }

```

Figura 9 - Mutation testing get

```

129     public ByteBuf get(long ledgerId, long entryId) {
130 1      lock.readLock().lock();
131
132     try {
133         // We need to check all the segments, starting from the current one and looking
134         // backward to minimize the
135         // checks for recently inserted entries
136         int size = cacheSegments.size();
137 3      for (int i = 0; i < size; i++) {
138 3          int segmentIdx = (currentSegmentIdx + (size - i)) % size;
139
140         LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
141 1      if (res != null) {
142         int entryOffset = (int) res.first;
143         int entryLen = (int) res.second;
144
145         ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
146         entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
147 1      return entry;
148     }
149     } finally {
150     }
151 1      lock.readLock().unlock();
152     }

```

Figura 10 – Coverage computeDigestAndPackageForSending

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• verifyDigest(long_ByteBuf_boolean)		100%		90%	1	6	0	22	0	1
• DigestManager(long_boolean_ByteBufAllocator)		100%	n/a	n/a	0	1	0	6	0	1
• verifyDigestAndReturnData(long_ByteBuf)		100%	n/a	n/a	0	1	0	3	0	1
• verifyDigest(long_ByteBuf)		100%	n/a	n/a	0	1	0	2	0	1
• static {...}		100%	n/a	n/a	0	1	0	1	0	1
• computeDigestAndPackageForSending(long_long_long_ByteBuf)		70%		37%	4	5	2	17	0	1
• instantiate(long_byte[]_DigestType_ByteBufAllocator_boolean)		43%		40%	3	5	3	6	0	1
• verifyDigestAndReturnLac(ByteBuf)		0%		0%	4	4	17	17	1	1
• computeDigestAndPackageForSendingLac(long)		0%		0%	2	2	8	8	1	1
• verifyDigestAndReturnLastConfirmed(ByteBuf)		0%	n/a	n/a	1	1	6	6	1	1
• generateMasterKey(byte[])		0%		0%	2	2	1	1	1	1
• update(byte[])		0%	n/a	n/a	1	1	2	2	1	1
• verifyDigest(ByteBuf)		0%	n/a	n/a	1	1	2	2	1	1
Total	222 of 468	52%	19 of 33	42%	19	31	41	93	6	13

Figura 11 – Analisi coverage computeDigestAndPackageForSending

```

102.     public ByteBufList computeDigestAndPackageForSending(long entryId, long lastAddConfirmed, long length,
103.         ByteBuf data) {
104.         ByteBuf headersBuffer;
105.         if (this.useV2Protocol) {
106.             headersBuffer = allocator.buffer(METADATA_LENGTH + macCodeLength);
107.         } else {
108.             headersBuffer = Unpooled.buffer(METADATA_LENGTH + macCodeLength);
109.         }
110.         headersBuffer.writeLong(ledgerId);
111.         headersBuffer.writeLong(entryId);
112.         headersBuffer.writeLong(lastAddConfirmed);
113.         headersBuffer.writeLong(length);
114.
115.         update(headersBuffer);
116.
117.         // don't unwrap slices
118.         final ByteBuf unwrapped = data.unwrap() != null && data.unwrap() instanceof CompositeByteBuf
119.             ? data.unwrap() : data;
120.         ReferenceCountUtil.retain(unwrapped);
121.         ReferenceCountUtil.release(data);
122.
123.         if (unwrapped instanceof CompositeByteBuf) {
124.             ((CompositeByteBuf) unwrapped).forEach(this::update);
125.         } else {
126.             update(unwrapped);
127.         }
128.         populateValueAndReset(headersBuffer);
129.         return ByteBufList.get(headersBuffer, unwrapped);
130.     }

```

Figura 12 – Coverage computeDigestAndPackageForSending (dopo miglioramento della test suite)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• verifyDigest(long_ByteBuf_boolean)		100%		90%	1	6	0	22	0	1
• DigestManager(long_boolean_ByteBufAllocator)		100%	n/a	n/a	0	1	0	6	0	1
• verifyDigestAndReturnData(long_ByteBuf)		100%	n/a	n/a	0	1	0	3	0	1
• verifyDigest(long_ByteBuf)		100%	n/a	n/a	0	1	0	2	0	1
• static {...}		100%	n/a	n/a	0	1	0	1	0	1
• computeDigestAndPackageForSending(long_long_long_ByteBuf)		96%		87%	1	5	0	17	0	1
• instantiate(long_byte[]_DigestType_ByteBufAllocator_boolean)		43%		40%	3	5	3	6	0	1
• verifyDigestAndReturnLac(ByteBuf)		0%		0%	4	4	17	17	1	1
• computeDigestAndPackageForSendingLac(long)		0%		0%	2	2	8	8	1	1
• verifyDigestAndReturnLastConfirmed(ByteBuf)		0%	n/a	n/a	1	1	6	6	1	1
• generateMasterKey(byte[])		0%		0%	2	2	1	1	1	1
• update(byte[])		0%	n/a	n/a	1	1	2	2	1	1
• verifyDigest(ByteBuf)		0%	n/a	n/a	1	1	2	2	1	1
Total	203 of 468	56%	15 of 33	54%	16	31	39	93	6	13

Figura 13 – Mutation testing computeDigestAndPackageForSending

```

102     public ByteBufList computeDigestAndPackageForSending(long entryId, long lastAddConfirmed, long length,
103         ByteBuf data) {
104         ByteBuf headersBuffer;
105 1   if (this.useV2Protocol) {
106 1   headersBuffer = allocator.buffer(METADATA_LENGTH + macCodeLength);
107     } else {
108 1   headersBuffer = Unpooled.buffer(METADATA_LENGTH + macCodeLength);
109     }
110     headersBuffer.writeLong(ledgerId);
111     headersBuffer.writeLong(entryId);
112     headersBuffer.writeLong(lastAddConfirmed);
113     headersBuffer.writeLong(length);
114
115 1   update(headersBuffer);
116
117     // don't unwrap slices
118 2   final ByteBuf unwrapped = data.unwrap() != null && data.unwrap() instanceof CompositeByteBuf
119     ? data.unwrap() : data;
120     ReferenceCountUtil.retain(unwrapped);
121     ReferenceCountUtil.release(data);
122
123 1   if (unwrapped instanceof CompositeByteBuf) {
124 1   ((CompositeByteBuf) unwrapped).forEach(this::update);
125     } else {
126 1   update(unwrapped);
127     }
128 1   populateValueAndReset(headersBuffer);
129 1   return ByteBufList.get(headersBuffer, unwrapped);
130 }

```

Figura 14 – Coverage verifyDigest

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cxty	Missed	Lines	Missed	Methods
• DigestManager(long, boolean, ByteBufAllocator)		100%		n/a	0	1	0	6	0	1
• verifyDigestAndReturnData(long, ByteBuf)		100%		n/a	0	1	0	3	0	1
• verifyDigest(long, ByteBuf)		100%		n/a	0	1	0	2	0	1
• static {...}		100%		n/a	0	1	0	1	0	1
• computeDigestAndPackageForSending(long, long, long, ByteBuf)		96%		87%	1	5	0	17	0	1
• verifyDigest(long, ByteBuf, boolean)		76%		70%	3	6	5	22	0	1
• instantiate(long, byte[], DigestType, ByteBufAllocator, boolean)		28%		20%	4	5	4	6	0	1
• verifyDigestAndReturnLac(ByteBuf)		0%		0%	4	4	17	17	1	1
• computeDigestAndPackageForSendingLac(long)		0%		0%	2	2	8	8	1	1
• verifyDigestAndReturnLastConfirmed(ByteBuf)		0%		n/a	1	1	6	6	1	1
• generateMasterKey(byte[])		0%		0%	2	2	1	1	1	1
• update(byte[])		0%		n/a	1	1	2	2	1	1
• verifyDigest(ByteBuf)		0%		n/a	1	1	2	2	1	1
Total	242 of 468	48%	18 of 33	45%	19	31	45	93	6	13

Figura 15 – Analisi coverage verifyDigest

```

163.     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
164.     throws BKDigestMatchException {
165.     ◆ if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
166.         logger.error("Data received is smaller than the minimum for this digest type. "
167.             + " Either the packet is corrupt, or the wrong digest is configured. "
168.             + " Digest type: {}, Packet Length: {}",
169.             this.getClass().getName(), dataReceived.readableBytes());
170.         throw new BKDigestMatchException();
171.     }
172.     update(dataReceived.slice(0, METADATA_LENGTH));
173.
174.     int offset = METADATA_LENGTH + macCodeLength;
175.     update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
176.
177.     ByteBuf digest = allocator.buffer(macCodeLength);
178.     populateValueAndReset(digest);
179.
180.     try {
181.     ◆ if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
182.         logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
183.         throw new BKDigestMatchException();
184.     }
185.     } finally {
186.         digest.release();
187.     }
188.
189.     long actualLedgerId = dataReceived.readLong();
190.     long actualEntryId = dataReceived.readLong();
191.
192.     ◆ if (actualLedgerId != ledgerId) {
193.         logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
194.             + actualLedgerId);
195.         throw new BKDigestMatchException();
196.     }
197.
198.     ◆ if (!skipEntryIdCheck && actualEntryId != entryId) {
199.         logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
200.             + actualEntryId);
201.         throw new BKDigestMatchException();
202.     }
203.
204. }

```

Figura 16 – Coverage verifyDigest (dopo miglioramento della test suite)

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Cxty	Missed	Lines	Missed	Methods
• verifyDigest(long, ByteBuf, boolean)		100%		90%	1	6	0	22	0	1
• DigestManager(long, boolean, ByteBufAllocator)		100%		n/a	0	1	0	6	0	1
• verifyDigestAndReturnData(long, ByteBuf)		100%		n/a	0	1	0	3	0	1
• verifyDigest(long, ByteBuf)		100%		n/a	0	1	0	2	0	1
• static {...}		100%		n/a	0	1	0	1	0	1
• computeDigestAndPackageForSending(long, long, long, ByteBuf)		96%		87%	1	5	0	17	0	1
• instantiate(long, byte[], DigestType, ByteBufAllocator, boolean)		43%		40%	3	5	3	6	0	1
• verifyDigestAndReturnLac(ByteBuf)		0%		0%	4	4	17	17	1	1
• computeDigestAndPackageForSendingLac(long)		0%		0%	2	2	8	8	1	1
• verifyDigestAndReturnLastConfirmed(ByteBuf)		0%		n/a	1	1	6	6	1	1
• generateMasterKey(byte[])		0%		0%	2	2	1	1	1	1
• update(byte[])		0%		n/a	1	1	2	2	1	1
• verifyDigest(ByteBuf)		0%		n/a	1	1	2	2	1	1
Total	203 of 468	56%	15 of 33	54%	16	31	39	93	6	13

Figura 17 – Analisi coverage verifyDigest (dopo miglioramento della test suite)

```

163.     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
164.     throws BKDigestMatchException {
165.     ◆ if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
166.         logger.error("Data received is smaller than the minimum for this digest type. "
167.             + " Either the packet is corrupt, or the wrong digest is configured. "
168.             + " Digest type: {}, Packet Length: {}",
169.             this.getClass().getName(), dataReceived.readableBytes());
170.         throw new BKDigestMatchException();
171.     }
172.     update(dataReceived.slice(0, METADATA_LENGTH));
173.
174.     int offset = METADATA_LENGTH + macCodeLength;
175.     update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
176.
177.     ByteBuf digest = allocator.buffer(macCodeLength);
178.     populateValueAndReset(digest);
179.
180.     try {
181.     ◆ if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
182.         logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
183.         throw new BKDigestMatchException();
184.     }
185.     } finally {
186.         digest.release();
187.     }
188.
189.     long actualLedgerId = dataReceived.readLong();
190.     long actualEntryId = dataReceived.readLong();
191.
192.     ◆ if (actualLedgerId != ledgerId) {
193.         logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
194.             + actualLedgerId);
195.         throw new BKDigestMatchException();
196.     }
197.
198.     ◆ if (!skipEntryIdCheck && actualEntryId != entryId) {
199.         logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
200.             + actualEntryId);
201.         throw new BKDigestMatchException();
202.     }
203.
204. }

```


Figura 18 – Mutation testing verifyDigest

```
163     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
164         throws BKDigestMatchException {
165 3       if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
166           logger.error("Data received is smaller than the minimum for this digest type. "
167               + " Either the packet is corrupt, or the wrong digest is configured. "
168               + " Digest type: {}, Packet Length: {}",
169               this.getClass().getName(), dataReceived.readableBytes());
170           throw new BKDigestMatchException();
171       }
172 1       update(dataReceived.slice(0, METADATA_LENGTH));
173
174 1       int offset = METADATA_LENGTH + macCodeLength;
175 2       update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
176
177       ByteBuf digest = allocator.buffer(macCodeLength);
178 1       populateValueAndReset(digest);
179
180       try {
181 1       if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
182           logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
183           throw new BKDigestMatchException();
184       }
185       } finally {
186           digest.release();
187       }
188
189       long actualLedgerId = dataReceived.readLong();
190       long actualEntryId = dataReceived.readLong();
191
192 1       if (actualLedgerId != ledgerId) {
193           logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
194               + actualLedgerId);
195           throw new BKDigestMatchException();
196       }
197
198 2       if (!skipEntryIdCheck && actualEntryId != entryId) {
199           logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
200               + actualEntryId);
201           throw new BKDigestMatchException();
202       }
203
204     }
```

Figura 19 – Mutation Testing verifyDigest (dopo rimozione mutante)

```

163     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
164         throws BKDigestMatchException {
165 3       if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
166         logger.error("Data received is smaller than the minimum for this digest type. "
167             + " Either the packet is corrupt, or the wrong digest is configured. "
168             + " Digest type: {}, Packet Length: {}",
169             this.getClass().getName(), dataReceived.readableBytes());
170         throw new BKDigestMatchException();
171     }
172 1       update(dataReceived.slice(0, METADATA_LENGTH));
173
174 1       int offset = METADATA_LENGTH + macCodeLength;
175 2       update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
176
177     ByteBuf digest = allocator.buffer(macCodeLength);
178 1       populateValueAndReset(digest);
179
180     try {
181 1       if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
182         logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
183         throw new BKDigestMatchException();
184     }
185     } finally {
186         digest.release();
187     }
188
189     long actualLedgerId = dataReceived.readLong();
190     long actualEntryId = dataReceived.readLong();
191
192 1       if (actualLedgerId != ledgerId) {
193         logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
194             + actualLedgerId);
195         throw new BKDigestMatchException();
196     }
197
198 2       if (!skipEntryIdCheck && actualEntryId != entryId) {
199         logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
200             + actualEntryId);
201         throw new BKDigestMatchException();
202     }
203
204 }

```

Figura 20 – Coverage compare

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
● encodeLong(long_byte[]_int)		100%		55%	8 10	0 31	0 1
● encodeInt(int_byte[]_int)		100%		75%	2 5	0 16	0 1
● encodeDouble(double_byte[]_int)		100%		n/a	0 1	0 12	0 1
● compareBytes(byte[]_int_int_byte[]_int_int)		100%		83%	1 4	0 8	0 1
● compare(byte[]_int_int_byte[]_int_int_Schema)		100%		n/a	0 1	0 6	0 1
● compare(byte[]_int_byte[]_int_Schema)		100%		n/a	0 1	0 1	0 1
● static {...}		100%		n/a	0 1	0 2	0 1
● compare(BinaryData.Decoders_Schema)		63%		48%	15 26	27 63	0 1
● hashCode(BinaryData.HashData_Schema)		0%		0%	18 18	28 28	1 1
● hashBytes(int_BinaryData.HashData_int_boolean)		0%		0%	4 4	11 11	1 1
● encodeFloat(float_byte[]_int)		0%		n/a	1 1	6 6	1 1
● hashCode(byte[]_int_int_Schema)		0%		n/a	1 1	5 5	1 1
● skipLong(byte[]_int)		0%		0%	2 2	2 2	1 1
● encodeBoolean(boolean_byte[]_int)		0%		0%	2 2	2 2	1 1
Total	373 of 1.005	62%	63 of 103	38%	54 77	81 193	6 14

Figura 21 – Analisi coverage compare

```

84. private static int compare(Decoders d, Schema schema) throws IOException {
85.     Decoder d1 = d.d1;
86.     Decoder d2 = d.d2;
87.     switch (schema.getType()) {
88.     case RECORD: {
89.         for (Field field : schema.getFields()) {
90.             if (field.order() == Field.Order.IGNORE) {
91.                 GenericDatumReader.skip(field.schema(), d1);
92.                 GenericDatumReader.skip(field.schema(), d2);
93.                 continue;
94.             }
95.             int c = compare(d, field.schema());
96.             if (c != 0) {
97.                 return (field.order() != Field.Order.DESENDING) ? c : -c;
98.             }
99.         }
100.        return 0;
101.    }
102.    case ENUM:
103.    case INT:
104.        return Integer.compare(d1.readInt(), d2.readInt());
105.    case LONG:
106.        return Long.compare(d1.readLong(), d2.readLong());
107.    case FLOAT:
108.        return Float.compare(d1.readFloat(), d2.readFloat());
109.    case DOUBLE:
110.        return Double.compare(d1.readDouble(), d2.readDouble());
111.    case BOOLEAN:
112.        return Boolean.compare(d1.readBoolean(), d2.readBoolean());
113.    case ARRAY: {
114.        long i = 0; // position in array
115.        long r1 = 0, r2 = 0; // remaining in current block
116.        long l1 = 0, l2 = 0; // total array length
117.        while (true) {
118.            if (r1 == 0) { // refill blocks(s)
119.                r1 = d1.readLong();
120.                if (r1 < 0) {
121.                    r1 = -r1;
122.                    d1.readLong();
123.                }
124.                l1 += r1;
125.            }
126.            if (r2 == 0) {
127.                r2 = d2.readLong();
128.                if (r2 < 0) {
129.                    r2 = -r2;
130.                    d2.readLong();
131.                }
132.                l2 += r2;
133.            }
134.            if (r1 == 0 || r2 == 0) // empty block: done
135.                return Long.compare(l1, l2);
136.            long l = Math.min(l1, l2);
137.            while (i < l) { // compare to end of block
138.                int c = compare(d, schema.getElementType());
139.                if (c != 0)
140.                    return c;
141.                i++;
142.                r1--;
143.                r2--;
144.            }
145.        }
146.    }

```

Figura 22 – Coverage compare (dopo miglioramento della test suite)





















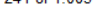
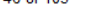
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● encodeLong(long_byte[],int)		100%		55%	8	10	0	31	0	1
● encodeInt(int_byte[],int)		100%		75%	2	5	0	16	0	1
● encodeDouble(double_byte[],int)		100%	n/a	n/a	0	1	0	12	0	1
● compareBytes(byte[],int,int_byte[],int,int)		100%		83%	1	4	0	8	0	1
● encodeFloat(float_byte[],int)		100%	n/a	n/a	0	1	0	6	0	1
● compare(byte[],int,int_byte[],int,int_Schema)		100%	n/a	n/a	0	1	0	6	0	1
● compare(byte[],int_byte[],int_Schema)		100%	n/a	n/a	0	1	0	1	0	1
● encodeBoolean(boolean_byte[],int)		100%		100%	0	2	0	2	0	1
● static {...}		100%	n/a	n/a	0	1	0	2	0	1
● compare(BinaryData_Decoders_Schema)		94%		87%	5	26	4	63	0	1
● hashCode(BinaryData_HashData_Schema)		0%		0%	18	18	28	28	1	1
● hashBytes(int_BinaryData_HashData_int_boolean)		0%		0%	4	4	11	11	1	1
● hashCode(byte[],int,int_Schema)		0%	n/a	n/a	1	1	5	5	1	1
● skipLong(byte[],int)		0%		0%	2	2	2	2	1	1
Total	241 of 1.005	76%	46 of 103	55%	41	77	50	193	4	14

Figura 23 – Analisi Coverage compare (dopo miglioramento della test suite)

```

84. private static int compare(Decoder d, Schema schema) throws IOException {
85.     Decoder d1 = d.d1;
86.     Decoder d2 = d.d2;
87.     switch (schema.getType()) {
88.     case RECORD: {
89.         for (Field field : schema.getFields()) {
90.             if (field.order() == Field.Order.IGNORE) {
91.                 GenericDatumReader.skip(field.schema(), d1);
92.                 GenericDatumReader.skip(field.schema(), d2);
93.                 continue;
94.             }
95.             int c = compare(d, field.schema());
96.             if (c != 0) {
97.                 return (field.order() != Field.Order.DESENDING) ? c : -c;
98.             }
99.         }
100.        return 0;
101.    }
102.    case ENUM:
103.    case INT:
104.        return Integer.compare(d1.readInt(), d2.readInt());
105.    case LONG:
106.        return Long.compare(d1.readLong(), d2.readLong());
107.    case FLOAT:
108.        return Float.compare(d1.readFloat(), d2.readFloat());
109.    case DOUBLE:
110.        return Double.compare(d1.readDouble(), d2.readDouble());
111.    case BOOLEAN:
112.        return Boolean.compare(d1.readBoolean(), d2.readBoolean());
113.    case ARRAY: {
114.        long i = 0; // position in array
115.        long r1 = 0, r2 = 0; // remaining in current block
116.        long l1 = 0, l2 = 0; // total array length
117.        while (true) {
118.            if (r1 == 0) { // refill blocks(a)
119.                r1 = d1.readLong();
120.                if (r1 < 0) {
121.                    r1 = -r1;
122.                    d1.readLong();
123.                }
124.                l1 += r1;
125.            }
126.            if (r2 == 0) {
127.                r2 = d2.readLong();
128.                if (r2 < 0) {
129.                    r2 = -r2;
130.                    d2.readLong();
131.                }
132.                l2 += r2;
133.            }
134.            if (r1 == 0 || r2 == 0) // empty block: done
135.                return Long.compare(l1, l2);
136.            long l = Math.min(l1, l2);
137.            while (i < l) { // compare to end of block
138.                int c = compare(d, schema.getElementType());
139.                if (c != 0)
140.                    return c;
141.                i++;
142.                r1--;
143.                r2--;
144.            }
145.        }
146.    }
147.    case MAP:
148.        throw new AvroRuntimeException("Can't compare maps!");
149.    case UNION: {
150.        int i1 = d1.readInt();
151.        int i2 = d2.readInt();
152.        int c = Integer.compare(i1, i2);
153.        return c == 0 ? compare(d, schema.getType().get(i1)) : c;
154.    }
155.    case FIXED: {
156.        int size = schema.getFixedSize();
157.        int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), size, d.d2.getBuf(), d.d2.getPos(), size);
158.        d.d1.skipFixed(size);
159.        d.d2.skipFixed(size);
160.        return c;
161.    }
162.    case STRING:
163.    case BYTES: {
164.        int l1 = d1.readInt();
165.        int l2 = d2.readInt();
166.        int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), l1, d.d2.getBuf(), d.d2.getPos(), l2);
167.        d.d1.skipFixed(l1);
168.        d.d2.skipFixed(l2);
169.        return c;
170.    }
171.    case NULL:
172.        return 0;
173.    default:
174.        throw new AvroRuntimeException("Unexpected schema to compare!");
175.    }
176. }

```

Figura 24 – Mutation testing compare

```

84 private static int compare(Decoders d, Schema schema) throws IOException {
85     Decoder d1 = d.d1;
86     Decoder d2 = d.d2;
87     switch (schema.getType()) {
88     case RECORD: {
89         for (Field field : schema.getFields()) {
90             if (field.order() == Field.Order.IGNORE) {
91                 GenericDatumReader.skip(field.schema(), d1);
92                 GenericDatumReader.skip(field.schema(), d2);
93                 continue;
94             }
95             int c = compare(d, field.schema());
96             if (c != 0) {
97                 return (field.order() != Field.Order.DESCEDING) ? c : -c;
98             }
99         }
100         return 0;
101     }
102     case ENUM:
103     case INT:
104         return Integer.compare(d1.readInt(), d2.readInt());
105     case LONG:
106         return Long.compare(d1.readLong(), d2.readLong());
107     case FLOAT:
108         return Float.compare(d1.readFloat(), d2.readFloat());
109     case DOUBLE:
110         return Double.compare(d1.readDouble(), d2.readDouble());
111     case BOOLEAN:
112         return Boolean.compare(d1.readBoolean(), d2.readBoolean());
113     case ARRAY: {
114         long i = 0; // position in array
115         long r1 = 0, r2 = 0; // remaining in current block
116         long l1 = 0, l2 = 0; // total array length
117         while (true) {
118             if (r1 == 0) { // refill blocks(s)
119                 r1 = d1.readLong();
120                 if (r1 < 0) {
121                     r1 = -r1;
122                     d1.readLong();
123                 }
124                 l1 += r1;
125             }
126             if (r2 == 0) {
127                 r2 = d2.readLong();
128                 if (r2 < 0) {
129                     r2 = -r2;
130                     d2.readLong();
131                 }
132                 l2 += r2;
133             }
134             if (r1 == 0 || r2 == 0) // empty block: done
135                 return Long.compare(l1, l2);
136             long l = Math.min(l1, l2);
137             while (i < l) { // compare to end of block
138                 int c = compare(d, schema.getElementType());
139                 if (c != 0)
140                     return c;
141                 i++;
142                 r1--;
143                 r2--;
144             }
145         }
146     }
147     case MAP:
148         throw new AvroRuntimeException("Can't compare maps!");
149     case UNION: {
150         int i1 = d1.readInt();
151         int i2 = d2.readInt();
152         int c = Integer.compare(i1, i2);
153         return c == 0 ? compare(d, schema.getType().get(i1)) : c;
154     }
155     case FIXED: {
156         int size = schema.getFixedSize();
157         int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), size, d.d2.getBuf(), d.d2.getPos(), size);
158         d.d1.skipFixed(size);
159         d.d2.skipFixed(size);
160         return c;
161     }
162     case STRING:
163     case BYTES: {
164         int l1 = d1.readInt();
165         int l2 = d2.readInt();
166         int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), l1, d.d2.getBuf(), d.d2.getPos(), l2);
167         d.d1.skipFixed(l1);
168         d.d2.skipFixed(l2);
169         return c;
170     }
171     case NULL:
172         return 0;
173     default:

```

Figura 25 – Mutation testing compare (dopo miglioramento della test suite)

```

84 private static int compare(Decoders d, Schema schema) throws IOException {
85     Decoder d1 = d.d1;
86     Decoder d2 = d.d2;
87     switch (schema.getType()) {
88     case RECORD: {
89         for (Field field : schema.getFields()) {
90             if (field.order() == Field.Order.IGNORE) {
91                 GenericDatumReader.skip(field.schema(), d1);
92                 GenericDatumReader.skip(field.schema(), d2);
93                 continue;
94             }
95             int c = compare(d, field.schema());
96             if (c != 0) {
97                 return (field.order() != Field.Order.DESCEENDING) ? c : -c;
98             }
99         }
100         return 0;
101     }
102     case ENUM:
103     case INT:
104         return Integer.compare(d1.readInt(), d2.readInt());
105     case LONG:
106         return Long.compare(d1.readLong(), d2.readLong());
107     case FLOAT:
108         return Float.compare(d1.readFloat(), d2.readFloat());
109     case DOUBLE:
110         return Double.compare(d1.readDouble(), d2.readDouble());
111     case BOOLEAN:
112         return Boolean.compare(d1.readBoolean(), d2.readBoolean());
113     case ARRAY: {
114         long i = 0; // position in array
115         long r1 = 0, r2 = 0; // remaining in current block
116         long l1 = 0, l2 = 0; // total array length
117         while (true) {
118             if (r1 == 0) { // refill blocks(s)
119                 r1 = d1.readLong();
120                 if (r1 < 0) {
121                     r1 = -r1;
122                     d1.readLong();
123                 }
124                 l1 += r1;
125             }
126             if (r2 == 0) {
127                 r2 = d2.readLong();
128                 if (r2 < 0) {
129                     r2 = -r2;
130                     d2.readLong();
131                 }
132                 l2 += r2;
133             }
134             if (r1 == 0 || r2 == 0) // empty block: done
135                 return Long.compare(l1, l2);
136             long l = Math.min(l1, l2);
137             while (i < l) { // compare to end of block
138                 int c = compare(d, schema.getElementType());
139                 if (c != 0)
140                     return c;
141                 i++;
142                 r1--;
143                 r2--;
144             }
145         }
146     }
147     case MAP:
148         throw new AvroRuntimeException("Can't compare maps!");
149     case UNION: {
150         int i1 = d1.readInt();
151         int i2 = d2.readInt();
152         int c = Integer.compare(i1, i2);
153         return c == 0 ? compare(d, schema.getTypes().get(i1)) : c;
154     }
155     case FIXED: {
156         int size = schema.getFixedSize();
157         int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), size, d.d2.getBuf(), d.d2.getPos(), size);
158         d.d1.skipFixed(size);
159         d.d2.skipFixed(size);
160         return c;
161     }
162     case STRING:
163     case BYTES: {
164         int l1 = d1.readInt();
165         int l2 = d2.readInt();
166         int c = compareBytes(d.d1.getBuf(), d.d1.getPos(), l1, d.d2.getBuf(), d.d2.getPos(), l2);
167         d.d1.skipFixed(l1);
168         d.d2.skipFixed(l2);
169         return c;
170     }
171     case NULL:
172         return 0;

```

Figura 26 – Coverage del metodo createSchema

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
• static {...}		100%		n/a	0	1	0
• SpecificData()		100%		n/a	0	1	0
• lambda\$getSchema\$1(Type)		100%		n/a	0	1	0
• isStringable(Class)		100%		n/a	0	1	0
• get()		100%		n/a	0	1	0
• getClassName(Schema)		87%		50%	3	4	1
• getSchema(Type)		86%		75%	1	3	0
• isEnum(Object)		81%		50%	2	3	0
• getSchemaName(Object)		81%		75%	1	3	1
• createSchema(Type, Map)		73%		72%	12	25	12
• getForSchema(Schema)		72%		66%	2	4	2
• getClass(Schema)		0%		0%	19	19	23
• SpecificData(ClassLoader)		0%		n/a	1	1	8
• getProtocol(Class)		0%		0%	2	2	8

Figura 27 – Análisi coverage del metodo createSchema

```

369.     protected Schema createSchema(java.lang.reflect.Type type, Map<String, Schema> names) {
370.         if (type instanceof Class && CharSequence.class.isAssignableFrom((Class) type))
371.             return Schema.create(Type.STRING);
372.         else if (type == ByteBuffer.class)
373.             return Schema.create(Type.BYTES);
374.         else if ((type == Integer.class) || (type == Integer.TYPE))
375.             return Schema.create(Type.INT);
376.         else if ((type == Long.class) || (type == Long.TYPE))
377.             return Schema.create(Type.LONG);
378.         else if ((type == Float.class) || (type == Float.TYPE))
379.             return Schema.create(Type.FLOAT);
380.         else if ((type == Double.class) || (type == Double.TYPE))
381.             return Schema.create(Type.DOUBLE);
382.         else if ((type == Boolean.class) || (type == Boolean.TYPE))
383.             return Schema.create(Type.BOOLEAN);
384.         else if ((type == Void.class) || (type == Void.TYPE))
385.             return Schema.create(Type.NULL);
386.         else if (type instanceof ParameterizedType) {
387.             ParameterizedType ptype = (ParameterizedType) type;
388.             Class raw = (Class) ptype.getRawType();
389.             java.lang.reflect.Type[] params = ptype.getActualTypeArguments();
390.             if (Collection.class.isAssignableFrom(raw)) { // array
391.                 if (params.length != 1)
392.                     throw new AvroTypeException("No array type specified.");
393.                 return Schema.createArray(createSchema(params[0], names));
394.             } else if (Map.class.isAssignableFrom(raw)) { // map
395.                 java.lang.reflect.Type key = params[0];
396.                 java.lang.reflect.Type value = params[1];
397.                 if (!(key instanceof Class && CharSequence.class.isAssignableFrom((Class) key)))
398.                     throw new AvroTypeException("Map key class not CharSequence: " + key);
399.                 return Schema.createMap(createSchema(value, names));
400.             } else {
401.                 return createSchema(raw, names);
402.             }
403.         } else if (type instanceof Class) { // class
404.             Class c = (Class) type;
405.             String fullName = c.getName();
406.             Schema schema = names.get(fullName);
407.             if (schema == null)
408.                 try {
409.                     schema = (Schema) (c.getDeclaredField("SCHEMA$").get(null));
410.                 } catch (NoSuchFieldException e) {
411.                     // HACK: schema mismatches class. maven shade plugin? try replacing.
412.                     schema = new Schema.Parser().
413.                         parse(schema.toString().replace(schema.getNamespace(), c.getPackage().getName()));
414.                 } catch (NoSuchFieldException e) {
415.                     throw new AvroRuntimeException("Not a Specific class: " + c);
416.                 } catch (IllegalAccessException e) {
417.                     throw new AvroRuntimeException(e);
418.                 }
419.             names.put(fullName, schema);
420.             return schema;
421.         }
422.         throw new AvroTypeException("Unknown type: " + type);
423.     }
424. }

```


Figura 28 – Coverage del metodo createSchema (dopo miglioramento della test suite)

```

32     public static String getGroupOwnerRealm(final String realmPath, final String groupKey) {
33 1     return realmPath + '@' + groupKey;
34     }

```

Figura 29 – Analisi coverage del metodo createSchema (dopo miglioramento della test suite)

```

369.     protected Schema createSchema(java.lang.reflect.Type type, Map<String, Schema> names) {
370.     ◆ if (type instanceof Class && CharSequence.class.isAssignableFrom((Class) type)) {
371.         return Schema.create(Type.STRING);
372.     ◆ else if (type == ByteBuffer.class)
373.         return Schema.create(Type.BYTES);
374.     ◆ else if ((type == Integer.class) || (type == Integer.TYPE))
375.         return Schema.create(Type.INT);
376.     ◆ else if ((type == Long.class) || (type == Long.TYPE))
377.         return Schema.create(Type.LONG);
378.     ◆ else if ((type == Float.class) || (type == Float.TYPE))
379.         return Schema.create(Type.FLOAT);
380.     ◆ else if ((type == Double.class) || (type == Double.TYPE))
381.         return Schema.create(Type.DOUBLE);
382.     ◆ else if ((type == Boolean.class) || (type == Boolean.TYPE))
383.         return Schema.create(Type.BOOLEAN);
384.     ◆ else if ((type == Void.class) || (type == Void.TYPE))
385.         return Schema.create(Type.NULL);
386.     ◆ else if (type instanceof ParameterizedType) {
387.         ParameterizedType ptype = (ParameterizedType) type;
388.         Class raw = (Class) ptype.getRawType();
389.         java.lang.reflect.Type[] params = ptype.getActualTypeArguments();
390.     ◆ if (Collection.class.isAssignableFrom(raw)) { // array
391.         if (params.length != 1)
392.             throw new AvroTypeException("No array type specified.");
393.         return Schema.createArray(createSchema(params[0], names));
394.     ◆ } else if (Map.class.isAssignableFrom(raw)) { // map
395.         java.lang.reflect.Type key = params[0];
396.         java.lang.reflect.Type value = params[1];
397.     ◆ if (!(key instanceof Class && CharSequence.class.isAssignableFrom((Class) key)))
398.         throw new AvroTypeException("Map key class not CharSequence: " + key);
399.         return Schema.createMap(createSchema(value, names));
400.     } else {
401.         return createSchema(raw, names);
402.     }
403.     ◆ } else if (type instanceof Class) { // class
404.         Class c = (Class) type;
405.         String fullName = c.getName();
406.         Schema schema = names.get(fullName);
407.     ◆ if (schema == null)
408.         try {
409.             schema = (Schema) (c.getDeclaredField("SCHEMA$").get(null));
410.         }
411.     ◆ if (!fullName.equals(getClassName(schema)))
412.         // HACK: schema mismatches class. maven shade plugin? try replacing.
413.         schema = new Schema.Parser()
414.             .parse(schema.toString().replace(schema.getNamespace(), c.getPackage().getName()));
415.     } catch (NoSuchFieldException e) {
416.         throw new AvroRuntimeException("Not a Specific class: " + c);
417.     } catch (IllegalAccessException e) {
418.         throw new AvroRuntimeException(e);
419.     }
420.     names.put(fullName, schema);
421.     return schema;
422. }
423. throw new AvroTypeException("Unknown type: " + type);
424. }
425.

```

Figura 30 – Mutation testing del metodo createSchema

```

369 protected Schema createSchema(java.lang.reflect.Type type, Map<String, Schema> names) {
370 2   if (type instanceof Class && CharSequence.class.isAssignableFrom((Class) type))
371 1       return Schema.create(Type.STRING);
372 1   else if (type == ByteBuffer.class)
373 1       return Schema.create(Type.BYTES);
374 2   else if ((type == Integer.class) || (type == Integer.TYPE))
375 1       return Schema.create(Type.INT);
376 2   else if ((type == Long.class) || (type == Long.TYPE))
377 1       return Schema.create(Type.LONG);
378 2   else if ((type == Float.class) || (type == Float.TYPE))
379 1       return Schema.create(Type.FLOAT);
380 2   else if ((type == Double.class) || (type == Double.TYPE))
381 1       return Schema.create(Type.DOUBLE);
382 2   else if ((type == Boolean.class) || (type == Boolean.TYPE))
383 1       return Schema.create(Type.BOOLEAN);
384 2   else if ((type == Void.class) || (type == Void.TYPE))
385 1       return Schema.create(Type.NULL);
386 1   else if (type instanceof ParameterizedType) {
387       ParameterizedType ptype = (ParameterizedType) type;
388       Class raw = (Class) ptype.getRawType();
389       java.lang.reflect.Type[] params = ptype.getActualTypeArguments();
390 1   if (Collection.class.isAssignableFrom(raw)) { // array
391 1       if (params.length != 1)
392           throw new AvroTypeException("No array type specified.");
393 1       return Schema.createArray(createSchema(params[0], names));
394 1   } else if (Map.class.isAssignableFrom(raw)) { // map
395       java.lang.reflect.Type key = params[0];
396       java.lang.reflect.Type value = params[1];
397 2   if (!(key instanceof Class && CharSequence.class.isAssignableFrom((Class) key)))
398       throw new AvroTypeException("Map key class not CharSequence: " + key);
399 1       return Schema.createMap(createSchema(value, names));
400   } else {
401 1       return createSchema(raw, names);
402   }
403 1   } else if (type instanceof Class) { // class
404       Class c = (Class) type;
405       String fullName = c.getName();
406       Schema schema = names.get(fullName);
407 1   if (schema == null)
408       try {
409           schema = (Schema) (c.getDeclaredField("SCHEMA$").get(null));
410       }
411 1   if (!fullName.equals(getClassName(schema)))
412       // HACK: schema mismatches class. maven shade plugin? try replacing.
413       schema = new Schema.Parser()
414           .parse(schema.toString().replace(schema.getNamespace(), c.getPackage().getName()));
415   } catch (NoSuchFieldException e) {
416       throw new AvroRuntimeException("Not a Specific class: " + c);
417   } catch (IllegalAccessException e) {
418       throw new AvroRuntimeException(e);
419   }
420   names.put(fullName, schema);
421 1   return schema;
422   }
423   throw new AvroTypeException("Unknown type: " + type);
424 }

```

```

99     public static Set<String> getEffective(final Set<String> allowedRealms, final String requestedRealm) {
100         Pair<Set<String>, Set<String>> normalized = normalize(allowedRealms);
101
102         Collection<String> requested = Arrays.asList(requestedRealm);
103
104         Set<String> effective = new HashSet<>();
105         effective.addAll(requested.stream().
106             filter(new StartsWithPredicate(normalized.getLeft())).collect(Collectors.toSet()));
107         effective.addAll(normalized.getLeft().stream().
108             filter(new StartsWithPredicate(requested)).collect(Collectors.toSet()));
109
110         // includes group ownership
111         effective.addAll(normalized.getRight());
112
113         // includes dynamic realms
114 1   if (allowedRealms != null) {
115         effective.addAll(allowedRealms.stream().filter(new DynRealmsPredicate()).collect(Collectors.toSet()));
116     }
117
118 1   return effective;
119 }

```