Report Software Testing Ingegneria del Software 2

Andrea Mazzucchi matr. 0286349

Introduzione

Il presente documento descrive le attività svolte sui progetti **BookKeeper**[1] e **OpenJPA**[2] dell'**Apache Software Foundation**. Verranno documentate in dettaglio tutte le operazioni effettuate, dalla prima build fino allo sviluppo dei test, evidenziando metodologie, strumenti utilizzati e risultati ottenuti.
Le classi su cui sono stati condotti i test sono state selezionate a partire dal dataset raccolto per il progetto riguardane la defect prediction tramite **machine learning**. Il criterio utilizzato per questa selezione è stato **LOC touched** (Lines of Code touched), ossia il numero di linee di codice aggiunte, rimosse o modificate in ciascuna release.
Data la necessità di sviluppare i test prevalentemente con un approccio **black-box**, sono stati presi in considerazione la documentazione delle classi e dei metodi, al fine di garantire una migliore copertura senza dipendere direttamente dall'implementazione.
Le metriche di copertura considerate sono la coverage delle istruzioni e delle branch ottenute tramite JaCoCo. Purtroppo non è stato possibile utilizzare Ba-dua per la dataflow coverage ad eccezione di una solo classe do OpenJPA.
Nel corso del report non verrà fatto riferimento a come sono stati scelti i valori poiché sono stati applicati i criteri di **category partition** e **boundary value analisys**.

BookKeeper

La prima operazione effettuata è stata la **build** del progetto (*mvn package*), partendo dal fork del repository ufficiale, seguendo le indicazioni fornite nella documentazione e includendo anche i test preesistenti.
Il primo ostacolo riscontrato è stata l'assenza del compilatore **C++** nell'ambiente di sviluppo. Dopo aver installato il compilatore, il processo di build è stato completato senza ulteriori problemi.
La versione di Java utilizzata è stata la **21** per una questione pratica: era già presente un file di configurazione per **GitHub Actions** dedicato alla build con questa versione.
Per quanto riguarda la configurazione del framework di CI, si è partiti dal workflow di Java 21 e sono stati rimossi tutti gli altri non necessari all'analisi corrente.
L'operazione successiva è stata la rimozione di tutti i test preesistenti del progetto per semplificare e velocizzare lo sviluppo. Sebbene sarebbe stato possibile escluderli selettivamente, si è preferita un'opzione più diretta e semplice, eliminandoli completamente.

Classi selezionate

Le classi selezionate per i test sono:

- **org.apache.bookkeeper.client.LedgerHandle**
- **org.apache.bookkeeper.client.BookKeeperAdmin**

Sebbene la documentazione generale non sia particolarmente dettagliata, **LedgerHandle** occupa un ruolo significativo, insieme alla sua variante **Advanced**, rendendola una scelta quasi obbligata per l'analisi e il testing.
La selezione di **BookKeeperAdmin**, invece, è stata guidata principalmente dai metodi documentati, che hanno fornito indicazioni utili alla scrittura dei test.

LedgerHandle

I metodi selezionati per i test sono:

- *public long addEntry (byte[] data)*
- *public CompletableFuture<Long> appendAsync (ByteBuf data)*
- *public void asyncAddEntry (final byte[] data, final AddCallback cb, final Object ctx)*
- *public long addEntry (byte[] data, int offset, int length)*
- *public long addEntry (final long entryId, byte[] data)*
- *public long addEntry (final long entryId, byte[] data, int offset, int length)*
- *public void asyncAddEntry (final long entryId, final byte[] data, final AddCallback cb, final Object ctx)*
- *public void asyncAddEntry (final long entryId, final byte[] data, final int offset, final int length, final AddCallback cb, final Object ctx)*
- *public Enumeration<LedgerEntry> readEntries (long firstEntry, long lastEntry)*

- *public CompletableFuture<LedgerEntries> readAsync (long firstEntry, long lastEntry)*
- *public Enumeration<LedgerEntry> batchReadEntries (long startEntry, int maxCount, long maxSize)*
- *public CompletableFuture<LedgerEntries> batchReadAsync (long startEntry, int maxCount, long maxSize)*
- *public long tryReadLastConfirmed ()*
- *public LedgerEntry readLastEntry ()*

L'elenco mostra chiaramente che sono stati selezionati solo metodi **pubblici**, scelta dettata dalla mancanza di documentazione per i metodi **privati** e per quelli visibili per il testing (*@VisibleForTesting*).
Un'assenza evidente è quella del **costruttore**, escluso in quanto **non pubblico**. Tuttavia, poiché la creazione di un'istanza di **LedgerHandle** viene effettuata attraverso il metodo **createLedger** della classe **BookKeeper**, è stato sviluppato anche un test per **createLedger**. Ovviamente **createLedger** chiama il costruttore durante la sua esecuzione**.**
Per la creazione dei test è stata recuperata la classe **BookKeeperClusterTest**.
Tutti i test sviluppati utilizzano la classe **BookKeeperClusterTest** per la creazione di un cluster BookKeeper, che viene inizializzato di base con 5 Bookie e un timeout di 30 secondi.

I seguenti tre metodi, sebbene apparentemente diversi, sono stati testati utilizzando gli stessi input per il parametro **data**:

- **public long addEntry (byte[] data),**
- **public CompletableFuture<Long> appendAsync (ByteBuf data),**
- **public void asyncAddEntry (final byte[] data, final AddCallback cb, final Object ctx)**

Gli input e i risultati attesi sono:

- **null -> FAIL;**
- **VALID_STRING -> PASS;**
- **EMPTY_STRING ->PASS.**

Dove VALID_STRING = "This is a test", convertito in un array di byte e EMPTY_STRING = "", convertito in un array di byte.
Il test è relativamente semplice: viene aggiunta un'entry al ledger e successivamente ne viene letto il contenuto, verificando che il valore restituito corrisponda a quello inserito.

public long addEntry (byte[] data, int offset, int length)
Questo metodo è una versione più articolata dei precedenti permetto tramite offset e length di indicare porzioni dell'array. Inoltre, un'analisi del codice ha rivelato che tutti e quattro i metodi menzionati chiamano:
public void asyncAddEntry(final byte[] data, final int offset, final int length, final AddCallback cb, final Object ctx).
Di conseguenza, i test sui tre metodi iniziali avrebbero potuto essere rimossi senza compromettere l'efficacia complessiva della suite di test, in quanto il loro comportamento viene comunque verificato.
La **Tabella 1** riporta i 18 input per il test.
I risultati mostrano una copertura del **100%** sia per le istruzioni che per le branch, dal calcolo sono state escluse le istruzioni e le branch relative al logging per il debug.

public long addEntry (final long entryId, byte[] data)
public long addEntry (final long entryId, byte[] data, int offset, int length)
public void asyncAddEntry (final long entryId, final byte[] data, final AddCallback cb, final Object ctx)
public void asyncAddEntry (final long entryId, final byte[] data, final int offset, final int length, final AddCallback cb, final Object ctx)
Questi quattro metodi non sono funzionanti per **LedgerHandle**, ma lo sono per **LedgerHandleAdv**.
Sono stati quindi inseriti test specifici per verificare che, tentando di aggiungere un'**entry** con valori leciti (già validati nei test precedenti), venga sempre sollevata un'**eccezione**.
Ovviamente, la **coverage** risulta essere del **100%**.

public Enumeration<LedgerEntry> readEntries (long firstEntry, long lastEntry)
public CompletableFuture<LedgerEntries> readAsync (long firstEntry, long lastEntry)
I parametri di input indicano il primo e l'ultimo id del range di entry che si vuole leggere.
Questi due metodi accettano lo stesso input ma seguono percorsi di esecuzione diversi fino alla chiamata di **asyncReadEntriesInternal**, che è stata identificata in un secondo momento.

Per il test, sono state inserite **NUM_ENTRIES = 5** entry nel **ledger** utilizzando **addEntry(byte[] data)**, metodo già verificato in precedenza, e attraverso i metodi in fase di test, è stato controllato che il numero di entry lette corrispondesse effettivamente a quelle che fanno parte.
La **Tabella 2** riporta gli 8 input per i test.
Per la coverage dei due metodi si ha sia per le istruzioni che per le branch il 100%., ma il metodo interno raggiunge l'80% per le istruzioni e il 50% per le branch. Da un'analisi del codice occorre inserire un test che tenti la lettura con il context client chiuso, ovvero chiudendo il client BookKeeper con cui è stato creato il ledger. Aggiungendo questo test anche per questo metodo si arriva ad una coverage del 100%.
Il metodo **asyncReadEntriesInternal** raggiunge il **100% di coverage** sia per le istruzioni che per le branch.

public Enumeration<LedgerEntry> batchReadEntries (long startEntry, int maxCount, long maxSize)
Questo metodo sembrava documentato, ma ha posto molti problemi nello sviluppo dei test poiché i parametri non risultavano molto chiari. Lo stesso problema si è riscontrato nella configurazione del **cluster** e del **client**..
Fortunatamente è disponibile la documentazione[3] relativa a questa API e si è riusciti a determinare che il client deve abilitarla tramite i parametri **BATCH_READ_ENABLED** e **USE_V2_WIRE_PROTOCOL**, inoltre l'Ack quorum e il Write quorum devo includere l'intero ensemble. Nonostante tutto ciò si è abbandonata la creazione dei test black-box come preventivato.
Nell'indicare i parametri di test oltre a **startEntry**, **maxCount** e **maxSize** si utilizza **batchEnabled**.
Per rendere più agevole il test, è stata modificata la classe **BookKeeperClusterTest**, introducendo:

- Un **costruttore dedicato** per gestire questa funzionalità.
- L'**accesso a una variabile**: nettyMaxFrameSizeBytes.

Il parametro **batchEnabled** viene utilizzato per indicare se la **lettura batch** è attivata. Se non lo è, il metodo utilizza **asyncReadEntriesInternal**, come nei test dei metodi precedenti.
Il parametro **maxCount** definisce il **numero massimo di entry** da leggere. Il numero effettivo di entry lette dipenderà da:

- **maxCount** e **maxSize**, se la lettura batch è abilitata.
- **maxCount** e **lastAddConfirmed**, se la lettura batch non è abilitata.

Comportamenti particolari:

- Se ci sono meno **entry** disponibili rispetto a **maxCount**, verranno lette tutte quelle presenti.
- Se **maxSize** è inferiore alla dimensione di una singola entry, verrà letta **solo una entry**.

Il test svolge le seguenti operazioni:

1. aggiunta delle entry al ledger;
2. lettura delle entry;
3. verifica del numero di entry letto.

La **Tabella 3** riporta gli input per i test.
Test 17 e 18: il problema potrebbe derivare dal controllo su firstEntry, che attualmente verifica solo se il valore è inferiore all'Id dell'ultima entry confermata, senza ulteriori validazioni.
Test 21, 22, 23, 24: manca completamente un controllo sul valore di maxCount.
Questi problemi indicano la necessità di una revisione del metodo per includere controlli più rigorosi su firstEntry e maxCount.
Prima di analizzare i dati relativi alla coverage occorre analizzare come il metodo si articola. La prima chiamata è a **asyncBatchReadEntries**, come tutti i metodi sincroni viene chiamata la variante asincrona.
asyncBatchReadEntries consiste di 3 branch:

- eccezione nel caso firstEntry sia superiore all'ultima entry confermata;
- **asyncReadEntriesInternal**, se batch non abilitato;
- **asyncBatchReadEntriesInternal**, se batch abilitato.

Per la coverage di istruzioni e branch abbiamo:

- **batchReadEntries 100% e n/a** (non ci sono branch);
- **asyncReadEntriesInternal**, per questo metodo si aveva già una coverage del 100% nei test precedenti;
- **asyncBatchReadEntriesInternal**, 81% e 50%, è necessario un test che tenti la lettura con il client chiuso.

Aggiunto il test si ha una coverage del 100%.

public CompletableFuture<LedgerEntries> batchReadAsync (long startEntry, int maxCount, long maxSize)
Per metodo le valgono le considerazioni relative alla configurazione e ai parametri fatte per il metodo precedente. Anche questo metodo esegue **asyncReadEntriesInternal**, nel caso la funzionalità batch non sia

disponibile, ma quando disponibile invoca **batchReadEntriesInternalAsync** invece di asyncBatchReadEntriesInternal.
Il test ricalca quello precedente e utilizza la **Tabella 3** per gli input.
In questo caso i test 17 e 18 vengono eseguiti restituendo il risultato atteso essendo presente il controllo mancante nel caso precedente.
Per i test 21, 22, 23, 24 si presenta il medesimo problema.
Anche in questo caso è necessario rivedere il metodo per inserire ulteriori controlli su maxCount.
Per la coverage abbiamo il 100% per entrambe le metriche. Per **batchReadEntriesInternalAsync** è 67% e 58%, anche in questo caso va introdoo un test con client chiuso, ma verranno ignorate gli altri branch poiché dai commenti al codice non è chiara l'intenzione degli sviluppatori. Con l'introduzione del nuovo test sia arriva al 73% per le istruzioni e al 66% per le branch.

public long tryReadLastConfirmed ()

Questo metodo restituisce l'id dell'ultima entry confermata non attendo le risposte da tutto il quorum, ma ignora tutte le risposte successive alla prima con un id maggiore dell'ultimo noto.
Sono stati effettuati 3 test:

1. sono state inserite NUM_ENTRIES entry e poi è stato verificato che l'id restituito corrisponda all'ultimo inserito;
2. stessa esecuzione della precedente, ma con una sola entry inserita;
3. esecuzione su un ledger dove senza alcuna entry, l'id restituito sarà *INVALID_ENTRY_ID* (-1).

La coverage del metodo è dell'87% per le istruzioni e del 75% per le branch.

public LedgerEntry readLastEntry ()

Questo metodo permette di ottenere l'ultima entry confermata.
Sono state eseguite le stesse operazioni del test precedente., ma se non sono presenti entry viene sollevta un eccezione invece che il ritorno di un valore invalido, ad esempio **null**.
Le coverage è del 100% per entrmbe le metriche.

BookKeeper createLedger(DigestType digestType, byte[] passwd)

Questo metodo è stato testato brevemente per individuare dei parametri validi per ottenere un istanza valida di LedgerHandle da utilizzare nei test precedenti. Infatti si è utilizzata la variante più semplice che richiede solo il digestType e una password.
Il test è anch'esso molto semplice, viene chiamato il metodo e viene verificato che il l'handle ottenuto permetta la scrittura.
La **tabella 4** riporta gli input del test.
La coverage del costruttore di LedgerHandle dopo l'esecuzione è del l'84% per le istruzione e del 37% per le branch.
Per tutti i test l'handler è stato creato utilizzando i parametri del test 1.

La coverage della classe è al 43% per le istruzioni e il 31% per le branch.

BookKeeperAdmin

I metodi selezionati per i test sono:

- *i costruttori;*
- *public LedgerHandle openLedger(final long lId)*
- *public Collection<BookieId> getAvailableBookies()*
- *public Collection<BookieId> getReadOnlyBookies()*
- *public void decommissionBookie(BookieId bookieAddress)*

I costruttori

Per testare il costruttore semplicemente si è utilizzato più volte utilizzando tutte le sue possibili varianti e verificando il suo funzionamento attraverso la chiamata a **public Collection<BookieId> getAllBookies()** e verificando che il numeor di Bookie corrisponda a quello con cui è stato inizializzato il cluster. Tra le varie possibilità un costruttore permette di creare un admin a partire da un client semplice quindi riprendendo i test precedenti s è testata la possibilità di utilizzare un client chiuso con ovviamente l'esito atteso e verificato di un eccezione sollevata.
Tutti i costruttori hanno una coverage del 100% per le istruzioni e non sono presenti branch.

public LedgerHandle openLedger(final long lId)
Questo metodo permette di ottenere un LedgerHandle già aperto fornendo solo l'id e ignorando la password. Per testare il funzionamento sono state eseguite le seguenti operazioni:

1. creazione del LedgerHandle;
2. aggiunta di una entry;
3. creazione di un admin;
4. apertura del ledger;
5. lettura dell'handle.

La coverage è del 100% e non sono presenti branch.

public Collection<BookieId> getAvailableBookies()
Il metodo permette di ottenere tutti gli id dei Bookie del cluster disponibili.
Il test segue il seguente ordine delle operazioni:

1. creazione dell'admin;
2. chiamata a getAllBookies() per ottenere l'elenco completo dei Bookie Id;
3. shudtown di uno dei Bookie;
4. chiamata al metodo testato;
5. Verifica che il numero di Bookie Id sia inferiore a quello di riferimento.

La coverage è del 100% e non sono presenti branch.

public Collection<BookieId> getReadOnlyBookies()
Il metodo permette di ottenere tutti gli id dei Bookie readOnly del cluster disponibili.
Il test segue il seguente ordine delle operazioni:

1. creazione dell'admin;
2. chiamata a getAllBookies() per ottenere l'elenco completo dei Bookie Id;
3. impostazione di uno dei Bookie in modalità readOnly;
6. chiamata al metodo testato;
7. Verifica che il numero di Bookie Id sia uguale ad 1.

La coverage è del 100% e non sono presenti branch.

public void decommissionBookie(BookieId bookieAddress)
decommissionBookie permette di replicare un Bookie non più disponibile in modo che non ci siano Ledger non correttamente replicati nel loro ensemble di riferimento. Le condizioni affinché il metodo funzioni correttamente sono:

- il cluster deve abilitare l'autoRecovery;
- il Bookie non deve essere in esecuzione.

Per permettere questo test è stato aggiunto un costruttore a BookKeeperClusterTest per abilitare e disabilitare l'autoRecovery.
Il test ha 2 input autoRecoveryEnabled, per impostare l'autoRecovery del cluster, e running, per indicare se il Bookie è o meno in esecuzione.
Il test si articola nei seguenti passaggi:

1. creazione di un LedgerHandle;
2. aggiunta di un entry all'handle;
3. shutdown di uno dei Bookie dell'ensemble dell'handle;
4. chiamata a decommissionBookie.

La **tabella 5** riporta gli input del test.
Utilizzando gli input del test 1 sono stati creati altri 2 test che testavano le seguenti situazioni:

- decommission di un Bookie readOnly;
- decommission di un Bookie senza LedgerHandle;

La coverage del metodo è del 64% per le istruzioni e del 50% per le branch.

La coverage della classe è al 20% per le istruzioni e il 10% per le branch.

Mutation Testing

L'esecuzione dell'intera suite di con il gruppo di default dei mutanti di PIT supera le 8 ore quindi si è optato per utilizzare le mutazioni singolarmente. I mutanti che causano timeout vengono considerati al pari dei killed.

CONDITIONALS_BOUNDARY
Per questa mutazione ci sono mutanti con coverage solo per LedgerHandle. Dei 44 mutanti generati, solo 22 vengono coperti dai test, 14 uccisi, 1 Timeout e 8 sopravvissuti. Indichiamo i mutanti con la rispettiva indice di linea:
210, 282, 289 interessarono il costruttore quindi verranno ignorati;
964, 958 riguardano batchReadEntriesInternalAsync e il controllo su maxSize, sono già stati testati tutti i valori limite per questi controlli e non è possibile causare un fallimento sul codice mutato;
983, 1161 i branch interessati sono stati già ignorati precedentemente nell'analisi;
1629 stessa boundary del due mutanti precedenti;
1645 riguarda un metodo per l'aggiornamento dell'attributo lastAddConfirmed, con la mutazione da > a >= viene incrementato un contatore di **hit** invece che di **miss**. Hit quando viene aggiornata lastAddConfirmed e miss quando viene chiamato inutilmente, tentare di creare un test che venga uccisi in queste condizioni viene ritenuto superfluo.

INCREMENTS
Non ci sono mutanti con coverage.

INVERT_NEGS
Non ci sono mutanti con coverage.

MATH
Per questa mutazione ci sono mutanti con coverage solo per LedgerHandle. 7 mutanti con coverage, 4 killed e 3 Timeout.

NEGATE_CONDITIONALS
L'esecuzione di questa mutazione impiega più di 2 ore quindi dato che il risultato di TestStrength per LedgerHandle è del 70% e per BookKeeperAdmin è dell'85% si termina l'analisi per BookKeeper.

OpenJPA
La prima operazione effettuata è stata la **build** del progetto con il comando *mvn clean install*, partendo dal fork della repository ufficiale e seguendo la documentazione, includendo anche i test preesistenti. Non ci sono stati particolari problemi una volta selezionata la versione corretta di **Java 11**.
Dopo la riuscita della build, come per **BookKeeper**, sono stati rimossi tutti i test per agevolare lo sviluppo. Anche la configurazione del framework di **CI** è stata semplice, poiché è bastato selezionare la versione corretta di Java.

Classi selezionate
Le classi selezionate per i test sono:
- **org.apache.openjpa.util.CacheMap**
- **org.apache.openjpa.util.ProxyManagerImpl**

La documentazione non è minimamente dettagliata per entrambe le classi infatti non si è rinunciato a qualsiasi tentativo di creare test da un punto di vista black-box

CacheMap
I metodi selezionati per i test sono:
- *il costruttore*
- *public boolean pin(Object key)*
- *public Object put(Object key, Object value)*
- *public Object remove(Object key)*

CacheMap è composta da tre mappe: **cacheMap**, **softMap** e **pinnedMap**. Quando cacheMap raggiunge la saturazione, le entry vengono spostate in softMap, mentre pinnedMap viene utilizzata per le chiavi contrassegnate come pinned.

Il costruttore
Il parametri del costruttori non sono di facile interpretazione quindi si è ridotto l'insieme di test a **lru**, **max**, e **size**. I parametri **load** e **concurrencyLevel** sono stati tralasciati.
Il test svolge le seguenti operazioni:

1. creazione della CacheMap;
2. verifica se LRU o meno;
3. verifica se la CacheMap ha il corretto valore di CacheSize ovvero **max;**
4. inserisce un numero di entry tale da saturare cacheMap.

La **tabella 6** riporta gli input del test e gli esiti attesi dei test.
La coverage del costruttore è del 100%.

public boolean pin(Object key)
Questo metodo permette di fissare una chive e il suo valore all'interno della CacheMap, gli oggetti pinned non sono considerati per quando riguarda la size massima e non sono mai rimossi implicitamente.
I vaolri di ritorno sono:

- true, se per la data chiave è presente una coppia key-value nella mappa;
- false, se non è presente un valore associato alla chiave.

È possibile fissare una chiave anche se non è presente alcun valore associato nella mappa.
Il test prende in input i valore **key**, **value** e **empty** riportati nella **tabella 7** insieme agli esiti attesi. **empty**, indica se la coppia chiave valore deve essere inserita durante il test.
Il test svolge le seguenti operazioni:

1. creazione della CacheMap;
2. inserimento, se richiesto, della coppia key-value;
3. esecuzione del metodo pin e verifica di key in pinnedMap;
4. unpin della key e verifica della rimozione.

La coverage del 80% delle istruzione e del 62% per le branch.
È necessario aggiungere all'insieme di test una verifca di pin per una key già nell'insieme pinned per raggiungere una coverage del 100%.

public Object put(Object key, Object value)
Questo metodo permette di inserire una coppia chiave-valore nella CacheMap. L'obiettivo di questo test è inserire almeno un valore in tutte le mappe e testare gli spostamenti di un coppia inserita tra cacheMap e softMap.
Vengono utilizzate 2 variabili booleane, pinned e softMap, per determinare le azioni da intraprendere durante il test. Vengono testate tutte le combinazioni possibili.
Il test svolge le seguenti operazioni:

1. creaziondi una CacheMap con size e max pari a 1 ;
2. viene inserito un valore con chiave 0 e oggetto Dummy come valore;
3. se pinned è true, viene fatto il pin della chiave 0;
4. se softMap è true, vengono inserite tante variabili in modo da saturare cacheMap e utilizzare softMap;
5. l'inserimento viene ripetuto per testare il comportamento di put nel caso di key già presenti.

La coverage è del 100% per entrambe le metriche.

public Object remove(Object key)
Questo metodo permette di rimuovere una coppia key-value da CacheMap. Nel caso key sia pinned viene rimosso solo il valore associato.
Per semplicità sono stati creati 2 test uno parametrico per uno dedicato alla rimozione dalla softMap.
Il test decidato alla softMap svolge le segenti operazioni:

1. creazioni di CacheMap con size e max pari a 1;
2. inserimenti di 3 elementi;
3. rimozione del primo elemento inserito, che si trova nella softMap;
4. verifica dell'elemento rimosso.

Il test parametrico utilizza 4 parametri: key, value, empty, pinned. Empty permette di inserire o meno la coppia key-value e pinned permette se fare o meno il pin della key.

Questo test svolge le seguenti operazioni:

1. creazioni di CacheMap;
2. se empty = true put della coppia key-value;
3. se pinned = true pin delle key;
4. rimozione della key;
5. verifiche della rimozione tenendo conto dei casi.

La **tabella 8** riporta gli input del test.
La coverage è del 100% per entrambe le metriche.

La coverage della classe è al 62% per le istruzioni e il 56% per le branch.

ProxyManagerImpl

I metodi selezionati sono:

- *public Object copyArray(Object orig)*
- *public Object copyCustom(Object orig)*

public Object copyArray(Object orig)

Questo metodo permette di ottenere un copia dell'array in input. In caso non sia possibile viene ritornato null.
Sono stati effettuati più test di copia:

- array di Integer vuoto;
- array di Integer con 3 elementi;
- array di Object, **{1, 2L, null, "Test"};**
- array di int;
- diverse stringhe;
- null.

Per tutti i test si è ottenuto il risultato atteso.
La coverage del metodo è del 100% per entrambe le metriche.

public Object copyCustom(Object orig)

Come il metodo precedente si ottiene una copia dell'input. È possibile ottenere la copia di: Proxy, PersistenceCapable, Collection, Map, Date, Calendar, JavaBean. In caso non sia possibile anche in questo caso si ottiene null.
Per questo test è stato necessario creare diversi oggetti dummy: Dummy, DummyProxy, Dummy PersistenceCapable, DummyBean.
La **tabella 9** riporta gli input del test e gli esiti attesi.
Il test è molto semplice viene chiamato il metodo e verificata la copia.
La coverage è del 100% per entrambe le metriche.

Mutation Testing

Per valutare la robustezza dei test, è stato eseguito il **Mutation Testing** utilizzando il gruppo di mutanti di default. Invece di eseguire direttamente i test, si è prima analizzato il comportamento dei metodi testati.
I risultati iniziali di **TestStrength** erano:

- **56%** per **CacheMap;**
- **69%** per **ProxyManagerImpl.**

CacheMap

Un primo miglioramento per i test consiste nell'utilizzo di **spy** e **verify** di **Mockito**,
Quindi nei test dei metodi pin, put and remove viene introdotto il controllo con verify sull'esecuzione di **writeLock** e **writeUnlock**.
Stessa cosa è stata ripetuta per **readLock** e **readUnlock** per i test che utilizzano i metodi size() e isEmpty().
Un ulteriore modifica ai test apportata è la verifica della size nel test **CacheMapPutTest** tramite assert.
Dopo queste modifiche la TestStrength è passata dal 56% al 67%.

ProxyManagerImpl

Nessun mutante che modifica i metedi in esame sopravvive. Il risultato di partenza è soddisfacente e data la difficoltà, e soprattutto la mancanza di documentazione, dei metodi interessati da mutanti che sopravvivono si tralascia un ulteriore evoluzione dell'insieme dei test.

Data Flow Coverage

Per CacheMap si era riusciti ad ottenere la data flow coverage tramite Ba-dua, ma con l'introduzioni di Mockito tutti i test che lo utilizzo causano diversi errori.

#	data	offset	length	Esito Atteso
1	VALID_STRING	-1	0	FAIL
2	VALID_STRING	-1	1	FAIL
3	VALID_STRING	0	len	PASS
4	VALID_STRING	0	len + 1	FAIL
5	VALID_STRING	0	len - 1	PASS
6	VALID_STRING	1	len	FAIL
7	VALID_STRING	1	len - 1	PASS
8	VALID_STRING	0	0	PASS
9	VALID_STRING	0	-1	FAIL
10	VALID_STRING	len	1	FAIL
11	VALID_STRING	len	0	PASS
12	VALID_STRING	len - 1	1	PASS
13	VALID_STRING	len - 1	0	PASS
14	VALID_STRING	len + 1	1	FAIL
15	VALID_STRING	len + 1	0	FAIL
16	EMPTY_STRING	0	0	PASS
17	EMPTY_STRING	0	1	FAIL
18	null	0	0	FAIL

Tabella 1: Input Test public long addEntry (byte[] data, int offset, int length)

#	firstEntry	lastEntry	expectedEntries	Esito Atteso
1	0	NUM_ENTRIES - 1	NUM_ENTRIES	PASS
2	1	NUM_ENTRIES - 3	NUM_ENTRIES - 3	PASS
3	-1	NUM_ENTRIES	NUM_ENTRIES	FAIL
4	-1	NUM_ENTRIES - 2	NUM_ENTRIES - 2	FAIL
5	0	NUM_ENTRIES	NUM_ENTRIES	FAIL
6	NUM_ENTRIES - 3	NUM_ENTRIES - 4	NUM_ENTRIES - 4	FAIL
7	NUM_ENTRIES - 1	NUM_ENTRIES - 1	NUM_ENTRIES -1	PASS
8	NUM_ENTRIES	NUM_ENTRIES	0	FAIL

Tabella 2: Input Test lettura entry

#	firstEntry	maxCount	MaxSize	batchEnabled	Esito Atteso
1	0	NUM_ENTRIES	Long.Max_VALUE	false	PASS
2	0	NUM_ENTRIES	Long.Max_VALUE	true	PASS

3	0	NUM_ENTRIES	nettyMaxFrameSizeBytes	false	PASS
4	0	NUM_ENTRIES	nettyMaxFrameSizeBytes	true	PASS
5	0	NUM_ENTRIES	NettyMaxFrameSizeBytes / 2	false	PASS
6	0	NUM_ENTRIES	NettyMaxFrameSizeBytes / 2	true	PASS
7	0	NUM_ENTRIES	1	false	PASS
8	0	NUM_ENTRIES	1	true	PASS
9	0	NUM_ENTRIES	0	false	PASS
10	0	NUM_ENTRIES	0	true	PASS
11	0	NUM_ENTRIES	-1	false	PASS
12	0	NUM_ENTRIES	-1	true	PASS
13	0	NUM_ENTRIES + 1	0	false	PASS
14	0	NUM_ENTRIES + 1	0	true	PASS
15	NUM_ENTRIES -1	NUM_ENTRIES	0	false	PASS
16	NUM_ENTRIES -1	NUM_ENTRIES	0	true	PASS
17	-1	NUM_ENTRIES	0	false	FAIL
18	-1	NUM_ENTRIES	0	true	FAIL
19	NUM_ENTRIES	NUM_ENTRIES	0	false	FAIL
20	NUM_ENTRIES	NUM_ENTRIES	0	true	FAIL
21	0	0	0	false	FAIL
22	0	0	0	true	FAIL
23	0	-1	0	false	FAIL
24	0	-1	0	true	FAIL
25	0	NUM_ENTRIES -1	0	false	PASS
26	0	NUM_ENTRIES -1	0	true	PASS

Tabella 3: Input Test batchReadEntries e batchReadAsync

#	DigestType	Password	Esito Atteso
1	DUMMMY	VALID_PASSWORD	PASS
2	CRC32	VALID_PASSWORD	PASS
3	CRC32C	VALID_PASSWORD	PASS
4	MAC	VALID_PASSWORD	PASS
5	MAC	EMPTY_PASSWORD	PASS
6	MAC	null	FAIL

Tabella 4: Input Test createLedger

#	autoRecoveryEnabled	running	Esito Atteso
1	true	false	PASS
2	false	false	FAIL
3	true	true	FAIL
4	False	true	FAIL

Tabella 5: Input Test decommissionBookie

#	lru	max	size	Esito Atteso
1	true	MAX	SIZE	PASS
2	false	MAX	SIZE	PASS
3	true	SIZE	MAX	PASS
4	false	SIZE	MAX	PASS
5	true	MAX	MAX	PASS
6	false	MAX	MAX	PASS
7	true	-1	SIZE	PASS
8	false	-1	SIZE	PASS
9	true	0	SIZE	FAIL
10	false	0	SIZE	FAIL
11	true	MAX	-1	PASS
12	false	MAX	-1	PASS
13	true	MAX	0	FAIL
14	false	MAX	0	FAIL

Tabella 6: Input Test costruttore CacheMap

#	key	value	empty	Esito Atteso
1	null	null	false	PASS
2	new Object()	new Object()	false	PASS
3	0	new Dummy(“Test” + 0, 0)	false	PASS
4	new Object()	null	false	PASS
5	null	new Object()	false	PASS
6	null	null	true	PASS
7	new Object()	new Object()	true	PASS
8	0	new Dummy(“Test” + 0, 0)	true	PASS
9	new Object()	null	true	PASS
10	null	new Object()	true	PASS

Tabella 7: Input Test pin

#	key	value	empty	pinned
1	null	null	true	true
2	null	null	true	false
3	null	null	false	true
4	null	null	false	false
5	new Object()	null	true	true
6	new Object()	null	true	false
7	new Object()	null	false	true
8	new Object()	null	false	false
9	new Object()	new Object()	true	true
10	new Object()	new Object()	true	false
11	new Object()	new Object()	false	true
12	new Object()	new Object()	false	false

Tabella 8: Input Test remove

#	orig	Esito Atteso
1	HashSet<Integer> intHashSet	PASS
2	LinkedList<String> stringList	PASS
3	ArrayList<Dummy> dummyList	PASS
4	HashMap<Integer> intMap	PASS
5	HashMap<String> stringMap	PASS
6	HashMap<Dummy> dummyMap	PASS
7	null	PASS
8	new Dummy("Test", 0)	FAIL
9	new DummyProxy(-1)	PASS
10	new DummyBean()	PASS
11	new DummyBean("Mario", 30, true)	PASS
12	new DummyPersistenceCapable()	FAIL
13	“Test”	FAIL
14	1	FAIL
15	new HashMap<>()	PASS
16	new ArrayList<>()	PASS
17	new Date(685098000000L)	PASS
18	New Date()	PASS
19	new GregorianCalendar(TimeZone.getTimeZone("Europe/Rome"))	PASS
20	new GregorianCalendar()	PASS

Tabella 9: Input Test copyCustom

Bibliografia

[1] https://bookkeeper.apache.org/
[2] https://openjpa.apache.org/
[3] https://bookkeeper.apache.org/bps/BP-62-new-API-for-batched-reads/#wire-protocol-changes