

Ingegneria Del Software 2

Progetto Parte De Angelis

Ezio Emanuele Ditella - 026766
Università Di Roma Tor Vergata

Abstract—L'insieme di metodi e pratiche definite dal *Software Quality Assurance* hanno assunto un ruolo cruciale nel processo di sviluppo software per monitorarne la sua qualità. Nel seguente documento ci concentreremo su una delle sue componenti, il *Software Testing*, applicandolo in un ambiente di *Continuous Integration*, *Continuous Deployment* e *Continuous Testing*.

I. INTRODUZIONE

Le applicazioni che saranno oggetto del processo di testing sono: *Apache Bookkeeper* ed *Apache Syncope*. Tali applicazioni sono state sviluppate in *java*, pertanto l'obiettivo del lavoro sarà costruire una *Test Suite* in grado di darci garanzie sufficienti sulla bontà delle Classi implementate.

Il processo di testing che è stato adottato si divide nei seguenti passi

- 1) Scelta delle classi da testare
- 2) Creazione di una Test Suite utilizzando il *Category Partition Method*
- 3) Raffinamento della Test Suite, basandoci su criteri di adeguatezza funzionale e strutturale
- 4) Applicazione del *Mutation Testing*

Prima di entrare nel dettaglio di ciascuna di tali fasi, è bene descrivere l'ambiente di sviluppo che si è utilizzato.

II. AMBIENTE DI SVILUPPO

I casi di test sono stati sviluppati usando i seguenti framework:

- **Junit** per la scrittura di *test di unità* facilmente ripetibili, modificabili ed integrabili con altri framework per l'applicazione dell'*automated testing*
- **Maven** come framework per configurare l'intero processo di *build* di un'applicazione, dall'installazione di dipendenze, all'esecuzione automatizzata dei casi di test
- **Git** come strumento di *version control* e per tenere traccia dell'avanzamento e del raffinamento della Test Suite
- **Travis** come strumento di *continuous integration*
- **Sonar Cloud** come servizio *online* per la valutazione della qualità del codice dell'applicazione, e **Jacoco**, **PitTest** strumenti *offline* usati per lo stesso scopo

Con questo tipo di ambiente ogni commit innescherà Travis, il quale innescherà il build dell'applicazione e quindi l'esecuzione dei test. Travis a sua volta innescherà SonarCloud per la valutazione della bontà del codice in termini di criteri di adeguatezza dei casi di test condotti.

Con questi framework garantiamo *Continuous* e *Automated Testing* e *Continuous Deployment*, consentendo di automatizzare ed integrare i processi del *Software Quality Assurance*

con il processo di Sviluppo e Deploy del software, e dunque applicare le pratiche descritte dal **DevOps**.

III. SCELTA DELLE CLASSI

Idealmete si vorrebbe **concentrare energie e budget** sulle classi per cui sia più probabile riscontrare dei difetti.

Per cercare di andare incontro a tale richiesta, è stato sviluppato un modello predittivo in grado di dire con quanta probabilità un certa Classe di un'applicazione, *Bookkeeper* o *Syncope*, sia difettosa.

Tra tutte le classi della singola applicazione, sono state scelte dunque quelle con più probabilità di avere un difetto. Questa operazione è stata fatta unendo i risultati prodotti da **WekaGUI** con i file dell'ultima release delle applicazioni usando uno script in *bash*. Uno snapshot dell'output prodotto dallo script è in Figura 23.

In particolare per *Bookkeeper* è stato scelto come modello predittivo il classificatore *RandomForest* senza *feature selection* ed *oversampling*. Mentre per *Syncope* è stato scelto il classificatore *NativeBayes* senza *feature selection* con *SMOTE*.¹ Nella Tabella I sono riportate le classi scelte.

Applicazione	Classe	% Difettività
Bookkeeper	LedgerHandle.java ¹	62%
	BookKeeperAdmin.java ²	87%
Syncope	BatchPayloadParser ³	N/A ⁵
	BatchPayloadLineReader ⁴	N/A ⁵

¹ bookkeeper-server/src/main/java/org/apache/bookkeeper/client/LedgerHandle.java

² bookkeeper-server/src/main/java/org/apache/bookkeeper/client/BookKeeperAdmin.java

³ common/idrepo/rest-api/src/test/java/org/apache/syncope/common/rest-api/batch/BatchPayloadParser.java

⁴ common/idrepo/rest-api/src/test/java/org/apache/syncope/common/rest-api/batch/BatchPayloadLineReader.java

⁵ Nessuna classe risultante dell'analisi esposta è stata ritenuta adeguata per l'esecuzione di test di unità

TABLE I

IV. CREAZIONE TEST SUITE

Pre la creazione di una Test Suite, per ogni metodo di ogni classe si è seguito il *Category Partition Method*.

Fissato un parametro del metodo da testare, si sono seguiti i seguenti passi:

- 1) Identificazione del dominio dei valori del parametro
- 2) Restringimento del dominio dei valori assumibili dal parametro, ove possibile, basandoci sulla documentazione dell'applicazione e sulle specifiche funzionali del metodo

¹Le motivazioni che hanno condotto a tale scelta sono giustificate alla conclusione di questo report <https://github.com/UltraLe/ProgettoISW2/report.pdf>

- 3) Partizionamento del dominio degli input in **classi di equivalenza**
- 4) Identificazione dei valori al bordo di ogni partizione, per poter applicare la **Bound Analysis**

Tale procedimento è stato iterato per ogni parametro dei ogni metodo. Ogni i-esimo passo di tale procedimento sarà riferito come CPM_i (eg. CPM_4 =Bound Analysis).

A questo punto per la creazione dei singoli casi di test multidimensionali si procederà in modo da avere una Test Suite **minimale**, ovvero si verificherà che ogni border value di ogni classe d'equivalenza dei parametri, compaia in almeno un caso di test.

Per ogni caso di test prodotto è stato inoltre specificato in che modo dovrebbe comportarsi il *SUT*.

Di seguito l'analisi delle classi dell'applicazione **BOOKKEEPER**.

A. *LedgerHandle*

Il client di Bookkeeper ha due ruoli fondamentali: creare e cancellare *ledgers*, ovvero sequenze di *entries*. La gestione dei ledgers avviene proprio tramite la classe *LedgerHandle*. Di tale classe sono stati testati metodi:

1) *getNumBookies()*: Ritorna il numero di bookie che posseggono parte di un ledger. Tale metodo non ha parametri quindi costruiamo due casi di test, uno in cui sono attivi 3 bookie, un altro in cui questi vengono arrestati. Ci aspettiamo che, anche dopo lo spegnimento dei bookie, il numero ritornato sia sempre pari a 3.

2) *addEntry(byte[] data)*: Il parametro *data* è un array di byte (CPM_1), le classi di equivalenza sono: valore diversi da null, uguali a null, array vuoti, array di taglia superiore alla taglia aspettata (CPM_3). Prenderemo dunque un array vuoto, uno con valore pari a null, ed uno di taglia pari a 100 MByte (CPM_4).

3) *addEntry(byte[] data, int offset, int length)*: Il parametro *offset* e *length* sono interi (CPM_1), partizionabili in due insiemi: uno contenente valori strettamente maggiori di zero, l'altro contenente valori minori di zero (CPM_3). Possiamo scegliere come border values per le partizioni di ciascun parametro i seguenti valori: -10, 0, 10 (CPM_4). Selezioniamo i Border Values per il parametro *data* come per il metodo precedente. Combiniamo ciascun valore per creare casi di test multidimensionale in modo da avere una test suite minimale.

A questo punto è stato verificato che il valore di ritorno fosse un valore maggiore di zero per i casi di test con combinazioni fattibili, e che sia stato gestito il caso di combinazioni non fattibili (eg. *offset* e/o *length* negativi).

4) *readEntries(long firstEntry, long lastEntry)*: Il parametro *firstEntry* è un intero (CPM_1) e si riferisce all'ID di un ledger. Possiamo partizionare tale parametro intero in una

partizione avente valori maggiori di zero, ed una aventi valori minori o uguali a zero (CPM_3). I Border Values per tale parametro saranno: -1, 0, 1 (CPM_4). Il ragoinamento è analogo per *lastEntry*. A questo punto si sono costruiti i casi di test multidimensionali sempre in modo minimale.

Prima di leggere le *entries*, è stato usato il metodo precedente per aggiungerne, ed è stato verificato che ciascuna *entry* aggiunta sia stata letta a partire da *firstEntry* fino a *lastEntry*. In modo simile è stato testato anche il metodo *readLastEntry()*.

B. *BookkeeperAdmin*

BookkeeperAdmin è la classe utilizzata per la gestione come amministratore del cluster di bookkeeper.

1) *CollectionBookieId getAvailableBookies()* : Metodo che controlla il numero di bookies disponibili. Non prende parametri in ingresso, Sono stati scelti i seguenti casi di test: uno in cui sono attivi e funzionanti 3 bookie, un altro in cui si interrompono, un altro in cui si parte da un insieme di bookie funzionanti e vengono effettuate 2 cancellazioni seguite da 2 recovery consecutive.

Per ogni caso di test è stato verificato che il numero di bookie attivo fosse corrispondente all'effettivo numero di bookie rimasti attivi.

2) *String formatEnsemble(List<BookieId> ensemble, <BookieId> bookiesSrc, char marker)*: Metodo che restituisce una stringa appendendo il carattere del parametro *marker* ai *BookieId* che fanno parte di un certo *ensemble*. I parametri *ensemble* e *bookiesSrc* sono una lista di *BookieId*, i cui border values possono essere: una lista vuota, una nulla, una lista con un insieme di *bookieId* (CPM_4). Il parametro *marker* può essere partizionato nei seguenti insiemi di caratteri: codificati ASCII, codificati UTF-8 (CPM_3). Dunque come border values prenderemo un carattere per ogni codifica (CPM_4).

Si vuole verificare che venga restituita una stringa con il carattere 'marker' al fianco di ogni *bookieId* definito da 'ensemble' che compare anche in 'bookiesSrc'. I casi di test per la costruzione della Test Suite iniziale sono stati definiti manualmente, senza effettuare il prodotto cartesiano tra i border values delle partizioni dei parametri.

3) *Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry)*: Il parametro *ledgerId* è un intero, riferito all'identificativo di un ledger, partizionabile in valori di ID leciti di ledger e valori non leciti, questi ultimi a loro volta partizionabili in positivi, e negativi (CPM_3). Sono stati scelti dunque come border values: un valore lecito di un ID di un ledger, e 2 valori non leciti, uno positivo l'altro negativo (CPM_4). La determinazione dei Border Values per i parametri *firstEntry* e *lastEntry* è stata fatta in modo analogo al metodo *readEntries* della classe *LedgerHandle*. È stato verificato che ciascuna *entry* aggiunta sia stata letta a partire

da `firstEntry` fino a `lastEntry`.

Di seguito l'analisi delle classi dell'applicazione **SYNCOPE**.

Le classi di Syncope che sono state testate sono utilizzate per gestire le *richieste batch*. Tali richieste permettono il raggruppamento di più operazioni (invocate tramite REST API) nel payload di una singola richiesta HTTP.

C. *BatchPayloadLineReader*

Classe usata per leggere richieste e risposte *batch*. I metodi di tale classe utilizzeranno come parametri una classe di tipo *InputStream* ed una di tipo *MediaType* (CPM_1), passati al costruttore della classe da testare. Il primo parametro è usato per estrarre il contenuto in formato Stringa della richiesta HTTP (CPM_2), pertanto sono state scelte le seguenti classi di partizioni (di stringhe) (CPM_3): una contenente richieste HTTP lecite, una contenente richieste improprie rispetto lo standard HTTP. Come border value è stato considerato un elemento arbitrario di ogni partizione (CPM_4).

Un ragionamento del tutto analogo è stato seguito per il parametro *MediaType*, il quale è usato per ottenere il tipo e sottotipo della richiesta HTTP, in formato stringa.

In particolare di tale classe è stato testato il metodo `List<BatchPayloadLine> read()`, il quale chiama ripetutamente il metodo privato `readLine()`. Il metodo è stato testato verificando che eventualmente terminasse, e che la lista ritornata contenga dei *BatchPayloadLine* leciti, ovvero una lista di linee della richiesta HTTP letta.

D. *BatchPayloadParser*

Una richiesta e/o risposta *batch* per poter essere correttamente gestita dovrà essere formattata in un certo modo. Sarà compito di questa classe parsare questi oggetti per poterli decomporre nelle singole operazioni che li costituiscono.

Il metodo pubblico di questa classe è `List<T> parse(InputStream in, MediaType multipartMixed, T template)>`, Con *T* una classe che estende un elemento di tipo *batch*, ovvero una richiesta o risposta.

Analogamente alla classe precedentemente discussa, i primi 2 parametri vengono utilizzati per estrarre rispettivamente: un payload HTTP e un parametro del suo header, il *boundary* ovvero una stringa usata per dividere una richiesta batch da un'altra. L'ultimo parametro è usata per comunicare alla classe se l'item batch si tratta di una richiesta o risposta.

Il **primo parametro** è stato partizionato in 2 classi di equivalenza (CPM_3): una contenente tutti i tipi di batch items non validi, ovvero la cui formattazione non rispetta le regole fornite dalle specifiche, ed una classe contenente batch item correttamente formattati. Per il **secondo parametro** le classi di equivalenza sono: una contenente tutti gli oggetti di tipo *MediaType* dalle quali è possibile estrarre un valore di *boundary* utilizzato nei batch items, e l'altra contenente

valori dai *boundary* non leciti. Per il **terzo parametro** le classi d'equivalenza sono: una contenente batch items di tipo richiesta, ed una di tipo risposta. Come *border values* (CPM_4) sono stati semplicemente prelevati un elemento di ogni partizione di ogni parametro.

La Test Suite iniziale per tale classe è stata creata in modo minimale, ovvero facendo in modo che ogni border value sia presente in almeno un caso di test.

Per ogni caso di test è stato verificato che il metodo *parse* sia stato in grado di ritornare, ove possibile², una lista di batch item i cui elementi sono corrispondenti al payload HTTP fornito al primo parametro.

V. RAFFINAMENTO DELLA TEST SUITE

Dopo aver costruito una versione iniziale della Test Suite, bisogna valutarne la bontà, in termini di criteri di adeguatezza strutturale e funzionale. In particolare il processo di raffinamento seguirà i seguenti passaggi:

- A Creazione di una serie di statement raccolti dalle specifiche delle applicazioni che spiegano il comportamento atteso dei metodi testati
- B Determinazione del numero degli statement coperti (già verificati) dalla Test Suite
- C Estrazione dei valori di Statement e Block Coverage dai report dei framework Jacoco e Sonar Cloud
- D Estensione della Test Suite in modo da coprire ogni statement prodotto al punto A
- E Estensione della Test Suite in modo da aumentare i valori discussi al punto B

A. Per la classe *LedgerHandle* dell'applicazione Bookkeeper:

- 1) `getNumBookies` ritorna il numero di Bookies attivi, uccisi o riavviati, che possiedono il ledger che ha invocato tale metodo
- 2) le entries sono aggiunte nei ledger sequenzialmente
- 3) le entries sono aggiunte nei ledger in al più una copia

Per la classe *BookKeeperAdmin*:

- 4) `getNumBookies` ritorna il numero di Bookie attivi di un cluster di bookkeeper
- 5) in `readEntries` se il parametro `lastEntry` è -1, leggerà tutte le entries nel ledger a partire da `firstEntry`

Per le classi *BatchPayloadLineReader* e *BatchPayloadParser* dell'applicazione Syncope:

- 6) una operazione in una richiesta batch per poter essere parsata correttamente deve attenersi alle specifiche riguardanti il *boundary* come definito nell'*RFC2046*

B. Dato l'insieme degli statement costruiti rappresentabile come [1, 2, 3, 4, 5, 6], la Test Suite soddisfa quelli in grassetto: [1, 2, 3, **4**, 5, 6]

C. Il build offline delle applicazioni con Maven innescherà la creazione del report del tool **Jacoco** mentre il build online

²per tutti i casi di test in cui i parametri forniti al metodo sono una terna lecita: (payload HTTP correttamente formattato, *boundary* contenuto nel payload HTTP, batch item di richiesta/risposta se il payload HTTP è un oggetto di richiesta/risposta)

eseguito con **TravisCI** innescherà l'analisi del sorgente fatta da **SonaCloud**. Le informazioni raccolte da Jacoco permettono di quantificare oggettivamente la **Statement Coverage** e **Block Coverage**³ della *Test Suite iniziale*, riportate nelle Figure 1, 2, 3, 4, mentre SonarCloud permette di quantificare (anche) la **Condition Coverage**, riportato nelle Figure 5, 6, 7, 8.

D. Per ricoprire gli statement 2 e 3, è necessario l'utilizzo sequenziale dei metodi `addEntryes` a `readEntryes`. Questa operazione è dunque fattibile solo dopo esserci assicurati che il metodo `readEntryes` abbia superato i test iniziali, assumendo che sia in grado di leggere nell'esatto ordine le entry aggiunte con `addEntryes`.

A questo punto sono state aggiunti dei casi di test in cui si è verificato che l'aggiunta delle entry sia sequenziale e univoca. Per ricoprire lo statent 5 si è banalmente aggiunto un caso di test che verificasse la lettura di tutte le entry aggiunte nel caso in cui `lastEntry` fosse -1.

Per ricoprire lo statement 6 sono stati aggiunti dei casi di test con delle richieste batch non lecite secondo lo standard definito dall' RFC2046, ed è stato verificato che la classe `BatchPayloadParser` generasse un'eccezione al tentativo di parsing di tali richieste.

E. Con l'obiettivo di aumentare i valori dei criteri di copertura come prima cosa si è passati dai casi di test minimali, a casi di test multidimensionali ottenuti facendo il prodotto cartesiano tra i border values di ciascuna partizione dei parametri dei metodi testati.

Dopodichè sono stati aggiunti dei casi di test 'ad-hoc' guardando l'implementazione del SUT, scegliendo i parametri in modo da

- soddisfare o meno condizioni composte/semplici per aumentare la Condition Coverage.
- raggiungere porzioni/blocchi di codice che non si erano raggiunti precedentemente, per aumentare la Statement e Block Coverage

Degli esempi dimostrativi del miglioramento ottenuto (in termini di copertura) sono mostrati in Fig. 9, 10, 11, 12, 13, 14.

Dopo queste operazioni si è riusciti ad aumentare il valore criteri di copertura, i cui nuovi valori calcolati con il tool **Jacoco** sono nelle Figure 15, 16, 17, 18. Mentre i nuovi valori estratti con l'ausilio di **SonarCloud** sono alle Figure 19, 20, 21, 22.

VI. MUTATION TESTING

Il *Mutation Testing* è una pratica usata per la valutazione della bontà della Test Suite molto interessante. Non rientra nè in criteri di adeguatezza funzionale nè strutturale. Difatti

l'idea è applicare delle *mutazioni* alle classi e verificare che i casi di test costruiti siano in grado di individuare, o *uccidere*, tale mutazioni.

L'operazione di mutation testing può considerarsi un successo nel caso in cui una Test Suite sia in grado di individuare ogni possibile tipo di mutazione. Essendo quest'operazione computazionalmente onerosa e time-consuming, e prendendo come riferimento l'insieme di mutanti offerti dal plugin *pitest*⁴, sono stati applicati in maniera sequenziale i seguenti mutanti:

- **Conditionals Boundary**: trasforma gli operatori `<` in `<=` e `<=` in `<` (analogamente con il segno `>`).
- **Arithmetic Operator Replacement**: scambia le operazioni `:` in `*` e `+` in `-` e viceversa.
- **Increments**: inibisce le operazioni di incremento e decremento da `i++`; o `(i--)` ad `i`;

I risultati ottenuti sono esposti in Tabella II.

VII. CONCLUSIONI E CONSIDERAZIONI

La fase del software testing nel processo del **Software Quality Assurance**, è una fase molto costosa e time consuming, e spesso bisogna andare in contro a trade off per trovare il giusto compromesso tra budget e ampiezza/bontà del processo stesso.

A tal riguardo, nell'ambito di questo progetto, gran parte del tempo è stato dedicato alle configurazioni dei framework discussi nella sezione *Ambiente Di Sviluppo* e alla configurazione dell'ambiente solo per poter effettuare il Build delle applicazioni analizzate. Questo ha portato alla scelta di sviluppare una test suite associata a criteri di copertura relativamente elevati, a scapito di doversi accontentare di valori di mutation coverage al di sotto del 90%.

Partendo da un ambiente pronto per applicare strategie DevOps, si sarebbero ottenute garanzie della qualità del software ben più elevate, in tempo e budget minore.

Ciò rispecchia in parte il ciclo di vita di un'azienda SW, la quale, secondo il modello **CMMI**, il superamento del primo livello si ottiene dallo sforzo 'eroico' dei dipendenti, nel senso che dovranno compiere maggiore effort nel muoversi in un ambiente inizialmente sregolato da standard e disorganizzato. Lo sforzo tuttavia servirà a portare l'azienda al **prima possibile** a livelli superiori, il che consentirà di utilizzare processi di sviluppo **standardizzati** e otterrà maggiore guadagno a parità dell'effort speso in passato.

L'**enfasi** va dunque nel non lanciarsi in un processo sregolato nella fase di testing, ma a studiare, comprendere, ed istruirsi sugli insiemi di standard da seguire, in termini di framework e/o piattaforme da usare, **come e perchè**. Anche se le fasi iniziali di questo apprendimento saranno poco proficue in termini di progresso nello sviluppo di un'applicazione, permetteranno poi, di concentrare in modo ottimale le risorse nel processo del **Software Quality Assurance**.

³In Jacoco la Statement coverage e Block Coverage sono i valori percentuali posizionati sulla destra dei parametri *Missed Instructions* e *Missed Branches*

⁴<https://pitest.org/quickstart/mutators/>

IMMAGINI REALTIVE ALLA TEST SUITE INIZIALE

LedgerHandle

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• asyncAddEntry(byte[], int, int, AsyncCallback.AddCallback, Object)		100%		66%
• updateLastConfirmed(long, long)		100%		100%
• drainPendingAddsAndAdjustLength()		100%		100%
• getNumBookies()		100%		100%
• readEntries(long, long)		100%		n/a
• asyncAddEntry(ByteBuf, AsyncCallback.AddCallback, Object)		100%		n/a
• errorOutPendingAdds(int, List)		100%		100%
• doAsyncCloseInternal(AsyncCallback.CloseCallback, Object, int)		100%		n/a
• lambda\$ensembleChangeLoop\$0(LedgerMetadata, Map.Entry)		100%		n/a
• addToLength(long)		100%		n/a
• addEntry(byte[])		100%		n/a
• getCurrentEnsemble()		100%		n/a
• getLedgerMetadata()		100%		n/a
• static { ... }		100%		n/a
• getId()		100%		n/a
• getLastAddConfirmed()		100%		n/a
• getVersionedLedgerMetadata()		100%		n/a
• getDistributionSchedule()		100%		n/a
• lambda\$ensembleChangeLoop\$1(Map, LedgerMetadata)		93%		50%
• isHandleWritable()		91%		50%
• readLastEntry()		86%		50%
• LedgerHandle(ClientContext, long, Versioned, BookKeeper.DigestType, byte[], EnumSet)		84%		37%
• readEntriesInternalAsync(long, long, boolean)		83%		50%
• addEntry(byte[], int, int)		82%		50%
• handleUnrecoverableErrorDuringAdd(int)		82%		50%
• asyncReadEntriesInternal(long, long, AsyncCallback.ReadCallback, Object, boolean)		80%		50%
• asyncReadExplicitLastConfirmed(AsyncCallback.ReadLastConfirmedCallback, Object)		70%		50%
• sendAddSuccessCallbacks()		69%		50%
• ensembleChangeLoop(List, Map)		69%		50%
• asyncReadLastConfirmed(AsyncCallback.ReadLastConfirmedCallback, Object)		66%		50%
• asyncReadEntries(long, long, AsyncCallback.ReadCallback, Object)		63%		66%
• doAsyncAddEntry(PendingAddOp)		62%		50%
• tearDownWriteHandleState()		58%		50%
• getWriteSetForReadOperation(long)		56%		50%
• handleBookieFailure(Map)		52%		33%
• maybeHandleDelayedWriteBookieFailure()		42%		50%
• initializeWriteHandleState()		34%		50%

Fig. 1. Statement e Block Coverage della classe LedgerHandle (Jacoco)

BookKeeperAdmin

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• formatEnsemble(List, Set, char)		100%		100%
• openLedgerNoRecovery(long)		100%		n/a
• BookKeeperAdmin(ClientConfiguration)		100%		n/a
• readEntries(long, long, long)		100%		75%
• BookKeeperAdmin(String)		100%		n/a
• static { ... }		100%		n/a
• getAvailableBookies()		100%		n/a

Fig. 2. Statement e Block Coverage della classe BookKeeperAdmin (Jacoco)

<i>BatchPayloadLineReader</i>					
Element	Missed Instructions	Cov.	Missed Branches	Cov.	
● readLine()		89%		87%	
● read()		100%		75%	
● BatchPayloadLineReader(InputStream, MediaType)		100%		n/a	
● fillBuffer()		100%		n/a	
● isBoundary(String)		100%		75%	
● close()		100%		n/a	
● static {...}		100%		n/a	

Fig. 3. Statement e Block Coverage della classe BatchPayloadLineReader (Jacoco)

<i>BatchPayloadParser</i>					
Element	Missed Instructions	Cov.	Missed Branches	Cov.	
● consumeHeaders(List, BatchItem)		85%		75%	
● split(List, String)		82%		64%	
● static {...}		100%		n/a	
● parse(InputStream, MediaType, BatchItem)		100%		n/a	
● lambda\$parse\$0(BatchItem, List)		100%		n/a	
● removeEndingCRLF(BatchPayloadLine)		100%		100%	
● removeEndingCRLFFromList(List)		100%		100%	
● consumeBlankLine(List)		100%		100%	

Fig. 4. Statement e Block Coverage della classe BatchPayloadParser (Jacoco)

LedgerHandle

Coverage	
Condition Coverage	24.6%
Conditions to Cover	264
Coverage	33.8%
Line Coverage	37.3%
Lines to Cover	705
Skipped Unit Tests	0
Uncovered Conditions	199
Uncovered Lines	442
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 5. Coverage della classe LedgerHandle (SonarCloud)

BookKeeperAdmin

Coverage	
Condition Coverage	7.5%
Conditions to Cover	214
Coverage	9.4%
Line Coverage	10.1%
Lines to Cover	595
Skipped Unit Tests	0
Uncovered Conditions	198
Uncovered Lines	535
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 6. Coverage della classe BookKeeperAdmin (SonarCloud)

BatchPayloadLineReader

Coverage	
Condition Coverage	85.7%
Conditions to Cover	42
Coverage	89.7%
Line Coverage	91.9%
Lines to Cover	74
Skipped Unit Tests	0
Uncovered Conditions	6
Uncovered Lines	6
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 7. Statement e Block Coverage della classe BatchPayloadLineReader (Jacoco)

BatchPayloadParser

Coverage	
Condition Coverage	76.0%
Conditions to Cover	50
Coverage	85.2%
Line Coverage	89.9%
Lines to Cover	99
Skipped Unit Tests	0
Uncovered Conditions	12
Uncovered Lines	10
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 8. Statement e Block Coverage della classe BatchPayloadParser (Jacoco)


```

1043.     public void asyncAddEntry(final byte[] data, final int offset, final int length,
1044.                               final AddCallback cb, final Object ctx) {
1045.         ◆ if (offset < 0 || length < 0 || (offset + length) > data.length) {
1046.             throw new ArrayIndexOutOfBoundsException(
1047.                 "Invalid values for offset(" + offset
1048.                 + ") or length(" + length + ")");
1049.         }
1050.
1051.         asyncAddEntry(Unpooled.wrappedBuffer(data, offset, length), cb, ctx);
1052.     }

```

```

1043.     public void asyncAddEntry(final byte[] data, final int offset, final int length,
1044.                               final AddCallback cb, final Object ctx) {
1045.         ◆ if (offset < 0 || length < 0 || (offset + length) > data.length) {
1046.             throw new ArrayIndexOutOfBoundsException(
1047.                 "Invalid values for offset(" + offset
1048.                 + ") or length(" + length + ")");
1049.         }
1050.
1051.         asyncAddEntry(Unpooled.wrappedBuffer(data, offset, length), cb, ctx);
1052.     }

```

Fig. 9. Miglioramento Coverage del metodo asyncAddEntries

```

678.     public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679.         // Little sanity check
680.         ◆ if (firstEntry < 0 || firstEntry > lastEntry) {
681.             LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682.                 ledgerId, firstEntry, lastEntry);
683.             cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684.             return;
685.         }
686.
687.         ◆ if (lastEntry > lastAddConfirmed) {
688.             LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689.                 ledgerId, firstEntry, lastEntry);
690.             cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691.             return;
692.         }
693.
694.         asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
695.     }

```

```

678.     public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679.         // Little sanity check
680.         ◆ if (firstEntry < 0 || firstEntry > lastEntry) {
681.             LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682.                 ledgerId, firstEntry, lastEntry);
683.             cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684.             return;
685.         }
686.
687.         ◆ if (lastEntry > lastAddConfirmed) {
688.             LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689.                 ledgerId, firstEntry, lastEntry);
690.             cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691.             return;
692.         }
693.
694.         asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
695.     }

```

Fig. 10. Miglioramento Coverage del metodo readAsyncEntries

363.	<code>public Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry)</code>
364.	<code>throws InterruptedException, BKException {</code>
365.	<code> checkArgument(ledgerId >= 0 && firstEntry >= 0);</code>
366.	<code> return new LedgerEntriesIterable(ledgerId, firstEntry, lastEntry);</code>
367.	<code>}</code>
363.	<code>public Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry)</code>
364.	<code>throws InterruptedException, BKException {</code>
365.	<code> checkArgument(ledgerId >= 0 && firstEntry >= 0);</code>
366.	<code> return new LedgerEntriesIterable(ledgerId, firstEntry, lastEntry);</code>
367.	<code>}</code>

Fig. 11. Miglioramento Coverage del metodo readEntries (BookKeeperAdmin)

```

100.  ◆      if (!foundLineEnd) {
101.          byte currentChar = buffer[offset++];
102.  ◆      if (!innerBuffer.hasRemaining()) {
103.          innerBuffer.flip();
104.          ByteBuffer tmp = ByteBuffer.allocate(innerBuffer.limit() * 2);
105.          tmp.put(innerBuffer);
106.          innerBuffer = tmp;
107.      }
108.      innerBuffer.put(currentChar);
109.
110.  ◆      if (currentChar == LF) {
111.          foundLineEnd = true;
112.  ◆      } else if (currentChar == CR) {
113.          foundLineEnd = true;
114.
115.          // Check next byte. Consume \n if available
116.          // Is buffer refill required?
117.  ◆      if (limit == offset) {
118.          fillBuffer();
119.      }
120.
121.          // Check if there is at least one character
122.  ◆      if (limit != EOF && buffer[offset] == LF) {
123.          innerBuffer.put(LF);
124.          offset++;
125.      }
126.  }
127. }
128. }

```

```

100.  ◆      if (!foundLineEnd) {
101.          byte currentChar = buffer[offset++];
102.  ◆      if (!innerBuffer.hasRemaining()) {
103.          innerBuffer.flip();
104.          ByteBuffer tmp = ByteBuffer.allocate(innerBuffer.limit() * 2);
105.          tmp.put(innerBuffer);
106.          innerBuffer = tmp;
107.      }
108.      innerBuffer.put(currentChar);
109.
110.  ◆      if (currentChar == LF) {
111.          foundLineEnd = true;
112.  ◆      } else if (currentChar == CR) {
113.          foundLineEnd = true;
114.
115.          // Check next byte. Consume \n if available
116.          // Is buffer refill required?
117.  ◆      if (limit == offset) {
118.          fillBuffer();
119.      }
120.
121.          // Check if there is at least one character
122.  ◆      if (limit != EOF && buffer[offset] == LF) {
123.          innerBuffer.put(LF);
124.          offset++;
125.      }
126.  }
127. }
128. }

```

Fig. 12. Miglioramento Coverage del metodo readLine (BatchPayloadLineReader)

```

72. private static List<List<BatchPayloadLine>> split(final List<BatchPayloadLine> lines, final String boundary) {
73.     List<List<BatchPayloadLine>> messageParts = new ArrayList<>();
74.     List<BatchPayloadLine> currentPart = new ArrayList<>();
75.     boolean isEndReached = false;
76.
77.     String quotedBoundary = Pattern.quote(boundary);
78.     Pattern boundaryDelimiterPattern = Pattern.compile("--" + quotedBoundary + "--\\s*");
79.     Pattern boundaryPattern = Pattern.compile("--" + quotedBoundary + "\\s*");
80.
81.     for (BatchPayloadLine line : lines) {
82.         if (boundaryDelimiterPattern.matcher(line.toString()).matches()) {
83.             removeEndingCRLFFromList(currentPart);
84.             messageParts.add(currentPart);
85.             isEndReached = true;
86.         } else if (boundaryPattern.matcher(line.toString()).matches()) {
87.             removeEndingCRLFFromList(currentPart);
88.             messageParts.add(currentPart);
89.             currentPart = new ArrayList<>();
90.         } else {
91.             currentPart.add(line);
92.         }
93.
94.         if (isEndReached) {
95.             break;
96.         }
97.     }
98.
99.     // Remove preamble
100.    if (!messageParts.isEmpty()) {
101.        messageParts.remove(0);
102.    }
103.
104.    if (!isEndReached) {
105.        int lineNumber = lines.isEmpty() ? 0 : lines.get(0).getLineNumber();
106.        throw new IllegalArgumentException("Missing close boundary delimiter around line " + lineNumber);
107.    }
108.
109.
110. private static List<List<BatchPayloadLine>> split(final List<BatchPayloadLine> lines, final String boundary) {
111.     List<List<BatchPayloadLine>> messageParts = new ArrayList<>();
112.     List<BatchPayloadLine> currentPart = new ArrayList<>();
113.     boolean isEndReached = false;
114.
115.     String quotedBoundary = Pattern.quote(boundary);
116.     Pattern boundaryDelimiterPattern = Pattern.compile("--" + quotedBoundary + "--\\s*");
117.     Pattern boundaryPattern = Pattern.compile("--" + quotedBoundary + "\\s*");
118.
119.     for (BatchPayloadLine line : lines) {
120.         if (boundaryDelimiterPattern.matcher(line.toString()).matches()) {
121.             removeEndingCRLFFromList(currentPart);
122.             messageParts.add(currentPart);
123.             isEndReached = true;
124.         } else if (boundaryPattern.matcher(line.toString()).matches()) {
125.             removeEndingCRLFFromList(currentPart);
126.             messageParts.add(currentPart);
127.             currentPart = new ArrayList<>();
128.         } else {
129.             currentPart.add(line);
130.         }
131.
132.         if (isEndReached) {
133.             break;
134.         }
135.     }
136.
137.     // Remove preamble
138.    if (!messageParts.isEmpty()) {
139.        messageParts.remove(0);
140.    }
141.
142.    if (!isEndReached) {
143.        int lineNumber = lines.isEmpty() ? 0 : lines.get(0).getLineNumber();
144.        throw new IllegalArgumentException("Missing close boundary delimiter around line " + lineNumber);
145.    }
146.
147.    return messageParts;
148. }

```

Fig. 13. Miglioramento Coverage del metodo split (BatchPayloadParser)

```

136.  ◆ if (ArrayUtils.contains(HTTP_METHODS, StringUtils.substringBefore(currentLine.toString(), " "))
137.      && item instanceof BatchRequestItem) {
138.
139.      BatchRequestItem bri = BatchRequestItem.class.cast(item);
140.      String[] parts = currentLine.toString().split(" ");
141.      bri.setMethod(parts[0]);
142.      String[] target = parts[1].split("\\?");
143.      bri.setRequestURI(target[0]);
144.  ◆ if (target.length > 1) {
145.      bri.setQueryString(target[1]);
146.  }
147.  ◆ } else if (item instanceof BatchResponseItem) {
148.      BatchResponseItem bri = BatchResponseItem.class.cast(item);
149.      try {
150.          bri.setStatus(Integer.valueOf(StringUtils.substringBefore(
151.              StringUtils.substringAfter(currentLine.toString(), " "), " ").trim()));
152.      } catch (NumberFormatException e) {
153.          LOG.error("Invalid value found in response for HTTP status", e);
154.      }
155.  }
156.  } else {
157.      Matcher headerMatcher = PATTERN_HEADER_LINE.matcher(currentLine.toString());
158.  ◆ if (headerMatcher.matches() && headerMatcher.groupCount() == 2) {
159.          itor.remove();
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.

```

Fig. 14. Miglioramento Coverage del metodo consumeHeaders (BatchPayloadParser)

LedgerHandle

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• asyncReadEntries(long, long, AsyncCallback.ReadCallback, Object)		100%		100%
• asyncAddEntry(byte[], int, int, AsyncCallback.AddCallback, Object)		100%		100%
• updateLastConfirmed(long, long)		100%		100%
• drainPendingAddsAndAdjustLength()		100%		100%
• getNumBookies()		100%		100%
• readEntries(long, long)		100%		n/a
• asyncAddEntry(ByteBuf, AsyncCallback.AddCallback, Object)		100%		n/a
• errorOutPendingAdds(int, List)		100%		100%
• doAsyncCloseInternal(AsyncCallback.CloseCallback, Object, int)		100%		n/a
• lambda\$ensembleChangeLoop\$0(LedgerMetadata, Map.Entry)		100%		n/a
• addToLength(long)		100%		n/a
• addEntry(byte[])		100%		n/a
• getCurrentEnsemble()		100%		n/a
• getLedgerMetadata()		100%		n/a
• static {...}		100%		n/a
• getId()		100%		n/a
• getLastAddConfirmed()		100%		n/a
• getVersionedLedgerMetadata()		100%		n/a
• getDistributionSchedule()		100%		n/a
• lambda\$ensembleChangeLoop\$1(Map, LedgerMetadata)		93%		50%
• isHandleWritable()		91%		50%
• readLastEntry()		86%		50%
• LedgerHandle(ClientContext, long, Versioned, BookKeeper.DigestType, byte[], EnumSet)		84%		37%
• readEntriesInternalAsync(long, long, boolean)		83%		50%
• addEntry(byte[], int, int)		82%		50%
• handleUnrecoverableErrorDuringAdd(int)		82%		50%
• asyncReadEntriesInternal(long, long, AsyncCallback.ReadCallback, Object, boolean)		80%		50%
• asyncReadExplicitLastConfirmed(AsyncCallback.ReadLastConfirmedCallback, Object)		70%		50%
• sendAddSuccessCallbacks()		69%		50%
• ensembleChangeLoop(List, Map)		69%		50%
• asyncReadLastConfirmed(AsyncCallback.ReadLastConfirmedCallback, Object)		66%		50%
• doAsyncAddEntry(PendingAddOp)		62%		50%
• tearDownWriteHandleState()		58%		50%
• getWriteSetForReadOperation(long)		56%		50%
• handleBookieFailure(Map)		52%		33%
• maybeHandleDelayedWriteBookieFailure()		42%		50%
• initializeWriteHandleState()		34%		50%

Fig. 15. Valori di Statement e Block Coverage della classe LedgerHandle

BookKeeperAdmin

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• formatEnsemble(List, Set, char)		100%		100%
• openLedgerNoRecovery(long)		100%		n/a
• BookKeeperAdmin(ClientConfiguration)		100%		n/a
• readEntries(long, long, long)		100%		100%
• BookKeeperAdmin(String)		100%		n/a
• static {...}		100%		n/a
• getAvailableBookies()		100%		n/a

Fig. 16. Valori di Statement e Block Coverage della classe BookKeeperAdmin

BatchPayloadLineReader











Element	Missed Instructions	Cov.	Missed Branches	Cov.
● readLine()		100%		100%
● read()		100%		100%
● BatchPayloadLineReader(InputStream, MediaType)		100%		n/a
● fillBuffer()		100%		n/a
● isBoundary(String)		100%		75%
● close()		100%		n/a
● static {...}		100%		n/a

Fig. 17. Valori di Statement e Block Coverage della classe BatchPayloadLineReader

BatchPayloadParser














Element	Missed Instructions	Cov.	Missed Branches	Cov.
● consumeHeaders(List, BatchItem)		97%		89%
● split(List, String)		100%		100%
● static {...}		100%		n/a
● parse(InputStream, MediaType, BatchItem)		100%		n/a
● lambda\$parse\$0(BatchItem, List)		100%		n/a
● removeEndingCRLF(BatchPayloadLine)		100%		100%
● removeEndingCRLFFromList(List)		100%		100%
● consumeBlankLine(List)		100%		100%

Fig. 18. Valori di Statement e Block Coverage della classe BatchPayloadParser

LedgerHandle

Coverage	
Condition Coverage	26.9%
Conditions to Cover	264
Coverage	35.6%
Line Coverage	38.9%
Lines to Cover	705
Skipped Unit Tests	0
Uncovered Conditions	193
Uncovered Lines	431
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 19. Valori di Coverage della classe LedgerHandle, calcolato da SonarCloud

BookKeeperAdmin

Coverage	
Condition Coverage	9.8%
Conditions to Cover	214
Coverage	10.5%
Line Coverage	10.8%
Lines to Cover	595
Skipped Unit Tests	0
Uncovered Conditions	193
Uncovered Lines	531
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 20. Valori di Coverage della classe BookKeeperAdmin, calcolato da SonarCloud

BatchPayloadLineReader

Coverage	
Condition Coverage	97.6%
Conditions to Cover	42
Coverage	98.3%
Line Coverage	98.6%
Lines to Cover	74
Skipped Unit Tests	0
Uncovered Conditions	1
Uncovered Lines	1
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 21. Valori di Coverage della classe BatchPayloadLineReader, calcolato da SonarCloud

BatchPayloadParser

Coverage	
Condition Coverage	94.0%
Conditions to Cover	50
Coverage	97.3%
Line Coverage	99.0%
Lines to Cover	99
Skipped Unit Tests	0
Uncovered Conditions	3
Uncovered Lines	1
Unit Test Errors	0
Unit Test Failures	0
Unit Test Success (%)	100%

Fig. 22. Valori di Coverage della classe BatchPayloadParser, calcolato da SonarCloud

RISULTATI MUTATION TESTING

Applicazione	Classe	MC ³	Mutanti					
			Conditionals		Boundary		Increments	
			AOR ¹		Applicato ²		Ucciso	
Bookkeeper	LedgerHandle	43%	11	5	0	0	3	1
	BookKeeperAdmin	75%	6	5	0	0	2	1
Syncope	BatchPayloadParser	80%	1	1	0	0	4	3
	BatchPayloadLineReader	53%	1	0	2	2	14	7

¹ Arithmetic Operator Replacement

² Applicato a dei metodi che sono stati testati

³ Mutation Coverage calcolata assumendo (controllando il report di pitest) un numero di mutanti *equivalenti* pari a zero

TABLE II

ALTRE IMMAGINI

```
bookkeeper-server/src/main/java/org/apache/bookkeeper/streaming/
LedgerInputStream.java
Yes at 0.62%
src/main/java/org/apache/distributedlog/util/Allocator.java
src/main/java/org/apache/distributedlog/impl/logsegment/
BKLogSegmentEntryStore.java
src/test/java/org/apache/distributedlog/admin/TestDLCK.java
bookkeeper-common/src/main/java/org/apache/bookkeeper/common/conf/
ComponentConfiguration.java
bookkeeper-server/src/main/java/org/apache/bookkeeper/client/
ReadOnlyLedgerHandle.java
bookkeeper-server/src/main/java/org/apache/bookkeeper/shims/zk/
package-info.java
bookkeeper-server/src/main/java/org/apache/bookkeeper/client/BKException.java
src/main/java/org/apache/distributedlog/api/AsyncLogReader.java
Yes at 0.63%
bookkeeper-server/src/main/java/org/apache/bookkeeper/discover/
RegistrationManager.java
bookkeeper-server/src/main/java/org/apache/bookkeeper/conf/package-info.java
src/test/java/org/apache/distributedlog/TestAppendOnlyStreamReader.java
src/main/java/org/apache/distributedlog/metadata/LogMetadataStore.java
src/main/java/org/apache/distributedlog/BKDistributedLogManager.java
Yes at 0.64%
bookkeeper-server/src/test/java/org/apache/bookkeeper/client/
TestDelayEnsembleChange.java
Yes at 0.65%
bookkeeper-common/src/main/java/org/apache/bookkeeper/common/stats/
BroadCastStatsLogger.java
bookkeeper-server/src/test/java/org/apache/bookkeeper/client/
MockBookKeeperTestCase.java
Yes at 0.66%
bookkeeper-server/src/test/java/org/apache/bookkeeper/client/
TestRackawareEnsemblePlacementPolicy.java
bookkeeper-server/src/main/java/org/apache/bookkeeper/conf/Configurable.java
src/main/java/org/apache/distributedlog/logsegment/TimeBasedRollingPolicy.java
src/main/java/org/apache/distributedlog/lock/LockAction.java
Yes at 0.67%
bookkeeper-server/src/test/java/org/apache/bookkeeper/bookie/EntryLogTest.java
```

Fig. 23. Classi di Bookkeeper divise per percentuale di difettosità (predetta)