

Report ISW2

Software Testing

Gabriele Quatrana 0306403

Sommario

1. Introduzione	1
2. JCS	1
2.1. RemovalTestUtil.java	1
2.2. ZeroSizeCacheUnitTest.java	2
2.3. Risultati	2
3. Bookkeeper	3
3.1. WriteCache	3
3.1.1. public boolean put(long ledgerId, long entryId, ByteBuf entry)	3
3.1.2. public ByteBuf get(long ledgerId, long entryId)	4
3.1.3. public ByteBuf getLastEntry(long ledgerId)	5
3.2. BookieImpl	5
3.2.1. public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)	5
3.2.2. public void recoveryAddEntry(ByteBuf entry, WriteCallback cb, Object ctx, byte[] masterKey)	6
3.2.3. public ByteBuf readEntry(long ledgerId, long entryId)	7
3.2.4. public long readLastAddConfirmed(long ledgerId)	8
4. Tajo	9
4.1. CatalogAdminClientImpl	9
4.1.1. public void createDatabase(String databaseName)	9
4.1.2. public void dropDatabase(String databaseName)	10
4.1.3. public TableDesc createExternalTable(String tableName, Schema schema, URI path, TableMeta meta, PartitionMethodDesc partitionMethodDesc)	11
4.2. SessionConnection	12
4.2.1. public Map<String, String> updateSessionVariables(Map<String, String> variables)	12
4.2.2. public String getSessionVariable(String varname)	13
5. Immagini	14
6. Riferimenti	29
6.1. JCS	29
6.2. Bookkeeper	29
6.3. Tajo	29

1. Introduzione

L'obiettivo di questo report è quello di descrivere tutte le attività svolte e le decisioni prese nell'attività di testing, riportando i risultati ottenuti per i progetti analizzati (**JCS**, **Bookkeeper** e **Tajo**). Per svolgere tali attività sono stati utilizzati dei programmi e framework di supporto. Come IDE è stato utilizzato **Eclipse**, per implementare i casi di test sono stati utilizzati **Junit4** e **Mockito**, per la gestione del ciclo di build è stato utilizzato **Maven**. Nei *pom* dei progetti sono stati definiti dei profili appositi per generare il report che descrive il livello di coverage basato su control-flow (tramite **Jacoco**) e quello che descrive il livello di robustezza dei test basato su mutation testing (tramite **Pit**). Tutte le attività sono state svolte in ottica di *Continuous Integration*, in modo tale da avere un riscontro relativo ai risultati ottenuti ad ogni commit. Lo strumento utilizzato per implementare il CI è stato **TravisCI**, che mette a disposizione un build server apposito sul quale è possibile eseguire, in un ambiente diverso da quello di sviluppo, il build del progetto e i test in modo automatico. Per analizzare il codice e visualizzare il report sulla coverage prodotto ad ogni commit, è stato utilizzato lo strumento **SonarCloud**, integrandolo nel processo di build **Maven**.

2. JCS

JCS (Java Caching System) è un sistema di caching distribuito scritto in linguaggio Java. L'obiettivo di questa attività è quello di scrivere dei test parametrici a partire dai test già presenti nel progetto considerato (JCS 1.3). Bisogna, quindi, convertire i test a disposizione, che sfruttano **Junit3**, in test parametrici che fanno utilizzo di **Junit4**. I test considerati sono `RemovalTestUtil.java` e `ZeroSizeCacheUnitTest.java`.

Per scrivere i nuovi casi di test è stato utilizzato principalmente un approccio Black-Box considerando i commenti presenti nelle classi di test del progetto.

Per convertire i test sono state effettuate diverse modifiche: è stato utilizzato il runner `JUnit4Parameterized.java`, è stato annotato il metodo di test con `@Test` ed è stato definito un nuovo metodo `getTestParameters()`, annotato con `@Parameters`, che si occupa di passare al costruttore della classe di test i parametri specificati. Attraverso questo metodo rendiamo più semplice la definizione di casi di test, in quanto il tester deve soltanto aggiungere una nuova entry nell'array ritornato dal metodo.

Per eseguire la configurazione dell'ambiente di test è stato definito un nuovo metodo apposito `setUp()`, annotato con `@Before`. Tale metodo si occupa di selezionare il corretto file di configurazione e di ottenere un'istanza di JCS da utilizzare nel test. Per eseguire la pulizia dell'ambiente è stato definito, invece, il metodo `cleanUp()`, annotato con `@After`. Tale metodo si occupa di pulire l'istanza di JCS utilizzata.

2.1. RemovalTestUtil.java

Questa classe comprende tre metodi di test che si occupano di collaudare l'inserimento, l'estrazione e la rimozione di un set di entry attraverso i metodi `put()`, `get()` e `remove()`.

I parametri di test individuati sono l'indice iniziale `start` e finale `end` utilizzati per definire il range di entry. Tali parametri sono stati derivati dagli argomenti dei metodi `runTestPutThenRemoveCategorical()` e `runPutInRange()`. Oltre ai due parametri precedenti, il metodo `runGetInRange()` utilizza il parametro booleano `check` per stabilire se una entry nulla è accettabile o meno.

Inizialmente sono stati utilizzati solo i test case presenti nel progetto all'interno della classe `ConcurrentRemovalLoadTest.java`:

- `runTestPutThenRemoveCategorical(): {0, 200} {601, 700} {701, 800}`
- `runPutInRange(): {300, 400} {401, 600}`
- `runGetInRange(): {0, 1000, false}`

Per eseguire i metodi di test separatamente con il proprio set di parametri è stata definita una enumerazione apposita ed è stato inserito un parametro che stabilisce il metodo da eseguire per ogni caso di test. Attraverso il metodo `assumeTrue()` di Junit è possibile stabilire quale metodo di test eseguire ad ogni nuovo run.

2.2. ZeroSizeCacheUnitTest.java

Questa classe comprende un solo metodo che si occupa di collaudare l'inserimento, l'estrazione e la rimozione di un range di entry attraverso i metodi `put()`, `get()` e `remove()`.

È stato individuato un solo parametro di test `items` che rappresenta il numero di entry utilizzate durante l'esecuzione dei test.

Inizialmente sono stati utilizzati solo i test case presenti nel progetto all'interno della classe `ZeroSizeCacheUnitTest.java`:

- `testPutGetRemove(): {20000}`

2.3. Risultati

Per analizzare il livello di coverage ottenuto con la test suite definita è stato necessario creare un nuovo profilo all'interno del `pom.xml` del progetto. Il profilo in questione implementa nel processo di build Maven il plugin **Jacoco**. Siccome il codice sorgente di JCS non è compreso nel nuovo progetto, **Jacoco** non può generare il report sulla coverage, per cui è necessario fornire al plugin una versione instrumentata di `jcs.jar`. Attraverso l'esecuzione del goal `prepare-agent`, **Jacoco** genera un report in formato `.exec`. Per analizzare il report è necessario convertirlo in un formato leggibile, come `html`, attraverso uno script `bash` eseguito nella fase `verify` di Maven mediante il plugin `exec-maven-plugin`.

La classe sperimentata dai test considerati è `org.apache.jcs.access.CacheAccess.java` che contiene i metodi `put()`, `get()` e `remove()`.

I risultati relativi alla coverage [\[Immagine 1\]](#), ottenuti a seguito dell'esecuzione dei casi di test considerati, mostrano che per i tre metodi è stato raggiunto un livello di coverage pari al **100%**: i casi di test originali presenti in JCS testano in modo esaustivo i metodi considerati. Non è stato quindi necessario espandere ulteriormente la test suite definita.

3. Bookkeeper

Bookkeeper è un servizio di storage tollerante ai guasti e a bassa latenza per carichi di lavoro real-time. Le classi scelte per l'attività di testing sono `WriteCache` e `BookieImpl`.

3.1. WriteCache

La classe `WriteCache` implementa una cache di scrittura, ovvero uno spazio di memoria suddiviso in vari segmenti, utilizzata dal `LedgerStorage` per accedere più rapidamente alle sue entry. Ogni entry viene identificata dall'ID del ledger e dall'ID che ha l'entry all'interno del ledger. I metodi sperimentati per questa classe sono stati `put()`, `get()` e `getLastEntry()` attraverso lo sviluppo di una test suite apposita.

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

Il metodo copia nella cache il contenuto della entry passata come input. All'interno della cache, l'entry viene identificata attraverso il suo `entryId` e il `ledgerId` del ledger in cui verrà scritta.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. I parametri `ledgerId` ed `entryId` sono degli identificativi, per cui ci si aspetta che dovrebbero essere non negativi. Per il parametro `entry` non è stato possibile fare nessuna particolare assunzione, per cui si è considerato un'istanza valida, una non valida ed una nulla: una istanza di `entry` è non valida, se la dimensione dell'entry stessa è maggiore della dimensione della cache o di un segmento della cache. La partizione in classi di equivalenza definita è stata la seguente:

- `ledgerId`: {<0, >=0}
- `entryId`: {<0, >=0}
- `entry`: {valid, invalid, null}

Per lo sviluppo dei casi di test è stato utilizzato un approccio unidimensionale, considerando quindi i vari parametri in modo indipendente e cercando di coprire con almeno un test tutte le classi di equivalenza definite. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {`ledgerId`, `entryId`, `entry`}
 - {1, 1, null} → null pointer exception
 - {-1, 0, invalid} → memory exception
 - {0, -1, valid} → illegal argument exception

Tutti i casi di test definiti hanno rispettato il comportamento atteso. La test suite così definita per il metodo `put()`, ha permesso di raggiungere una statement coverage pari al **96%** ed una branch coverage pari al **62%** [\[Immagine 2\]](#). Inizialmente, attraverso un approccio **Black-Box**, sono stati definiti due ulteriori casi di test:

- {0, 0, valid} → no exception
- {-1, 0, valid} → illegal argument exception

La nuova test suite non ha portato a dei miglioramenti per la coverage; quindi, si è passati ad un approccio **White-Box**. Analizzando il report fornito da **Jacoco** per il metodo `put()` [\[Immagine 3\]](#), si è visto che non vengono coperti tre branch. Dai commenti presenti nel codice si può vedere che uno dei branch a riga **172** (`!lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)`) non può essere coperto in nessun modo e di conseguenza nemmeno il codice a riga **175** verrà coperto.

Per il branch a riga **167** è stato introdotto un nuovo caso di test tale per cui `currentLastEntryId > entryId`. Il test introdotto consiste nell'andare ad inserire due entry in modo tale che l'ID della seconda sia minore dell'ID della prima.

Per il branch a riga **150** è stato introdotto un nuovo caso di test tale per cui `maxSegmentSize - localOffset < size`. Il test introdotto consiste nell'andare ad inserire due entry in modo tale che la seconda

non possa far parte dello stesso segmento della prima: le entry hanno una dimensione pari alla metà della dimensione del segmento più uno.

Il report sulla coverage dopo i miglioramenti apportati alla classe di test, mostra una statement coverage del **98%** e un branch coverage del **87%** [\[Immagine 4\]](#) [\[Immagine 5\]](#).

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 6\]](#) si può vedere che l'unica mutazione non rilevata dalla test suite è quella a riga **147**, in cui l'addizione presente nella condizione viene sostituita da una sottrazione. Per migliorare la robustezza della test suite sono stati introdotti due ulteriori casi di test:

- `{1, 1, entry_segment_size}`: la entry in questo caso ha una dimensione pari a quella dei segmenti.
- `{1, 1, entry_empty}`: la entry in questo caso ha una dimensione nulla.

Tuttavia, anche a seguito dell'introduzione di questi nuovi casi di test, non è stato possibile rilevare la mutazione. Per migliorare la robustezza della test suite sarebbe necessario uno studio approfondito del codice e del suo funzionamento; quindi, si è deciso di mantenere inalterata la test suite sviluppata senza riservare ulteriore effort per il metodo `put()`.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **98%** e una branch coverage pari al **87%**, ed è risultata molto robusta in quanto non è riuscita a rilevare solo **una** mutazione.

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

Il metodo restituisce in output la entry identificata dall'ID del ledger e dall'ID della entry stessa e salvata nella cache.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome:

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

Per testare il funzionamento del metodo, in fase di configurazione viene inserita nella cache una entry, tramite il metodo `put()`, con `ledgerId` pari a 1 ed `entryId` pari a 1. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{ledgerId, entryId}`
 - `{1, 1}` → cache hit
 - `{2, 2}` → cache miss
 - `{-1, 0}` → illegal argument exception
 - `{0, -1}` → illegal argument exception

Il caso di test `{0, -1}` non ha rispettato il comportamento atteso. Infatti, non ha sollevato nessuna eccezione, quindi il parametro `entryId` può assumere anche valori negativi. A seguito di questa osservazione, è stato modificato il comportamento del caso di test che non dovrà sollevare nessuna eccezione.

La test suite così definita ha permesso di raggiungere il **100%** sia di statement coverage che di branch coverage: risulta quindi essere una test suite adeguata per il metodo considerato [\[Immagine 2\]](#) [\[Immagine 7\]](#).

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 8\]](#) si può vedere che l'unica mutazione non rilevata dalla test suite è quella a riga **194**, in cui lo shift verso destra viene sostituito da uno shift verso sinistra. Per migliorare la robustezza della test suite sarebbe necessario uno studio approfondito del codice e del suo funzionamento; quindi, si è deciso di mantenere inalterata la test suite sviluppata senza riservare ulteriore effort per il metodo `get()`.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **100%** e un branch coverage pari al **100%**, ed è risultata molto robusta in quanto non è riuscita a rilevare solo una mutazione.

3.1.3. public ByteBuf getLastEntry(long ledgerId)

Il metodo restituisce in output l'ultima entry presente nel ledger identificato dal suo ID e salvata nella cache.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dal parametro, basandosi sulla semantica del nome:

- ledgerId: {<0, >=0}

Per testare il funzionamento del metodo, in fase di configurazione viene inserita nella cache una entry, tramite il metodo `put()`, con `ledgerId` pari a 1 ed `entryId` pari a 1. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {ledgerId}
 - {1} → cache hit
 - {0} → cache miss
 - {-1} → illegal argument exception

La test suite così definita ha permesso di raggiungere il **100%** sia di statement coverage che di branch coverage: risulta quindi essere una test suite adeguata per il metodo considerato [\[Immagine 2\]](#) [\[Immagine 9\]](#).

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 10\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **100%** e un branch coverage pari al **100%**, ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni.

3.2. BookieImpl

La classe `BookieImpl` implementa un **Bookie**, ovvero un nodo di archiviazione di **Bookkeeper**, in cui vengono salvate le entry. I **Bookie** sono server di archiviazione il cui scopo è quello di mantenere il contenuto dei **Ledger**. I metodi sperimentati per questa classe sono stati `addEntry()`, `recoveryAddEntry()`, `readEntry()` e `readLastAddConfirmed()` attraverso lo sviluppo di una test suite apposita.

3.2.1. public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)

Il metodo aggiunge una entry all'interno del **Bookie** considerato. La entry da inserire nel **Bookie** ha la seguente forma:

- long + long + byte[] → ledgerId + entryId + data

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dal parametro, basandosi sulla semantica del nome. Per i parametri `entry`, `cb` e `ctx` sono state considerate un'istanza valida, una non valida ed una nulla. Una entry non è valida se almeno uno degli ID che contiene è negativo. Il parametro `masterKey` è una chiave rappresentata da un array di byte, per cui ci si aspetta che dovrebbe essere non vuoto e che non dovrebbe superare una lunghezza massima. La partizione in classi di equivalenza definita inizialmente è stata la seguente:

- entry: {valid, invalid, null}
- ackBeforeSync: {true, false}
- cb: {valid, invalid, null}

- `ctx: {valid, invalid, null}`
- `masterKey: {valid, invalid, empty}`

Attraverso un approccio **Black-Box** non è stato possibile definire delle istanze appropriate per la partizione `{invalid}` di `cb`, `ctx` e `masterKey`. Analizzando il codice attraverso un approccio **White-Box** si è scoperto che non è possibile definire delle istanze non valide per il metodo considerato. La nuova partizione in classi di equivalenza è la seguente:

- `entry: {valid, invalid, null}`
- `ackBeforeSync: {true, false}`
- `cb: {valid, null}`
- `ctx: {valid, null}`
- `masterKey: {valid, empty}`

Per lo sviluppo dei casi di test è stato utilizzato un approccio unidimensionale, considerando quindi i vari parametri in modo indipendente e cercando di coprire con almeno un test tutte le classi di equivalenza definite. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{entry, ackBeforeSync, cb, ctx, masterKey}`
 - `{valid, false, null, null, valid}` → `no exception`
 - `{null, true, valid, null, empty}` → `null pointer exception`
 - `{invalid, false, null, valid, valid}` → `illegal argument exception`

Tutti i casi di test hanno rispettato il comportamento atteso. La test suite così definita per il metodo `addEntry()`, ha permesso di raggiungere una statement coverage pari al **83%** ed una branch coverage pari al **75%** [Immagine 11].

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [Immagine 12] fornito da **Jacoco** per il metodo `addEntry()`, si è visto che non viene coperto un branch e un blocco catch.

Il branch a riga **1408** (`handle.isFenced() == true`) può essere coperto andando ad eseguire il “fence” del ledger. È stato, quindi, sviluppato un nuovo caso di test che esegue il metodo `fenceLedger()` sul ledger in cui viene inserita la entry.

Il blocco catch a riga **1416** può essere coperto sollevando l'eccezione `NoWritableLedgerDirException`. Per implementare il caso di test descritto sarebbe necessario uno studio approfondito del codice; quindi, si è deciso di non riservare ulteriore effort per il blocco considerato.

Il report sulla coverage dopo i miglioramenti apportati alla classe di test mostra una statement coverage del **87%** ed una branch coverage del **100%** [Immagine 13] [Immagine 14].

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [Immagine 15] si può vedere che non tutte le mutazioni sono state rilevate. In particolare, non sono state rilevate 5 mutazioni. Analizzando il codice si è scoperto che le mutazioni introdotte sono relative a delle funzioni di logging, per cui non sono state prese in considerazione.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **87%** ed una branch coverage pari al **100%**, ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni di interesse.

3.2.2. `public void recoveryAddEntry(ByteBuf entry, WriteCallback cb, Object ctx, byte[] masterKey)`

Il metodo permette di aggiungere una entry all'interno del **Bookie** considerato, anche se il **Ledger** in cui viene inserita la entry è nello stato “fenced”. Quando un **Ledger** è “fenced” non è possibile scrivere in esso con il metodo `addEntry()`.

La partizione in classi di equivalenza definita è uguale a quella del metodo analizzato in precedenza:

- `entry`: {valid, invalid, null}
- `cb`: {valid, null}
- `ctx`: {valid, null}
- `masterKey`: {valid, empty}

Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {`entry`, `ackBeforeSync`, `cb`, `ctx`, `masterKey`}
 - {valid, null, null, valid} → no exception
 - {null, valid, null, empty} → null pointer exception
 - {invalid, null, valid, valid} → illegal argument exception

Tutti i casi di test hanno rispettato il comportamento atteso. La test suite così definita per il metodo `recoveryAddEntry()`, ha permesso di raggiungere una statement coverage pari al **86%** ed una branch coverage pari al **100%** [Immagine 11].

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [Immagine 16] fornito da **Jacoco** per il metodo `recoveryAddEntry()`, si è visto che non viene coperto un solo blocco catch.

Il blocco catch a riga **1329** può essere coperto sollevando l'eccezione `NoWritableLedgerDirException`. Per implementare il caso di test descritto sarebbe necessario uno studio approfondito del codice; quindi, si è deciso di non riservare ulteriore effort per il blocco considerato.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [Immagine 17] si può vedere che tutte le mutazioni sono state rilevate, tranne quelle relative alle funzioni di logging, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **86%** ed una branch coverage pari al **100%**, ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni di interesse.

3.2.3. `public ByteBuf readEntry(long ledgerId, long entryId)`

Il metodo restituisce in output la entry identificata da `entryId` e salvata nel ledger identificato da `ledgerId`, mantenuto a sua volta dal bookie considerato.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. I parametri `ledgerId` ed `entryId` sono degli identificativi, per cui ci si aspetta che dovrebbero essere non negativi:

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

Per testare il funzionamento del metodo, in fase di configurazione viene inserita nella cache una entry, tramite il metodo `addEntry()`, con `ledgerId` pari a 1 ed `entryId` pari a 1. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {`ledgerId`, `entryId`}
 - {1, 1} → hit
 - {1, 2} → entry miss
 - {2, 1} → ledger miss
 - {0, -1} → illegal argument exception
 - {-1, 0} → illegal argument exception

I casi di test {0, -1} e {-1, 0} non hanno rispettato il comportamento atteso. Infatti, hanno sollevato l'eccezione `NoLedgerException`, quindi i parametri `ledgerId` e `entryId` possono assumere anche valori negativi. A seguito di questa osservazione, sono stati rimossi questi due casi di test.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **89%** ed una branch coverage pari al **75%** [Immagine 11].

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [Immagine 18] fornito da **Jacoco** per il metodo `readEntry()`, si è visto che non viene coperto un solo branch.

Il branch a riga **1454** (`LOG.isTraceEnabled() == true`) è relativo ad operazioni di logging, di conseguenza non è stato speso effort aggiuntivo per coprire tale codice.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [Immagine 19] si può vedere che tutte le mutazioni sono state rilevate, tranne quelle relative alle funzioni di logging, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **89%** ed una branch coverage pari al **75%**, ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni di interesse.

3.2.4. public long readLastAddConfirmed(long ledgerId)

Il metodo restituisce in output l'ultimo *long* salvato all'interno del ledger identificato dal proprio ID e mantenuto a sua volta dal bookie considerato.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. Il parametro `ledgerId` è un identificativo, per cui ci si aspetta che dovrebbe essere non negativo:

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

Per testare il funzionamento del metodo, in fase di configurazione viene inserita nella cache una entry, tramite il metodo `addEntry()`, con `ledgerId` pari a 1 ed `entryId` pari a 1. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{ledgerId}`
 - `{1}` → hit
 - `{0}` → ledger miss
 - `{-1}` → illegal argument exception

Il caso di test {-1} non ha rispettato il comportamento atteso. Infatti, ha sollevato l'eccezione `NoLedgerException`, quindi il parametro `ledgerId` può assumere anche valori negativi. A seguito di questa osservazione, è stato rimosso questo caso di test.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **100%** ed una branch coverage nulla in quanto il metodo non contiene nessun branch [Immagine 11] [Immagine 20].

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [Immagine 21] si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **100%** ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni.

4. Tajo

Tajo è un framework di elaborazione dati relazionale e distribuito, progettato per fornire una bassa latenza. Le classi scelte per l'attività di testing sono `CatalogAdminClientImpl` e `SessionConnection`.

4.1. CatalogAdminClientImpl

La classe `CatalogAdminClientImpl` implementa un client di interazione con **Tajo**, attraverso il quale è possibile eseguire delle query SQL tramite dei metodi appositi. I metodi sperimentati per questa classe sono stati `createDatabase()`, `dropDatabase()` e `createExternalTable()` attraverso lo sviluppo di una test suite apposita.

4.1.1. public void createDatabase(String databaseName)

Questo metodo permette di creare un nuovo database il cui nome viene specificato in input.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. Il parametro `databaseName` è una stringa che indica il nome del database da creare. Inizialmente, la partizione in classi di equivalenza definita è stata la seguente:

- `databaseName: {valid_string, invalid_string, empty_string}`

Come stringhe non valide sono state considerate stringhe composte da caratteri speciali, come `@` o `!`. Per lo sviluppo dei casi di test è stato utilizzato un approccio unidimensionale, considerando quindi i vari parametri in modo indipendente e cercando di coprire con almeno un test tutte le classi di equivalenza definite. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{databaseName}`
 - `{valid_string}` → no exception
 - `{invalid_string}` → illegal argument exception
 - `{empty_string}` → illegal argument exception

I casi di test `{invalid_string}` e `{empty_string}` non hanno rispettato il comportamento atteso. Infatti, non hanno sollevato nessuna eccezione, quindi il parametro `databaseName` può contenere qualsiasi carattere ed essere una stringa vuota. A seguito di questa osservazione, sono stati eliminati questi casi di test.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **76%** ed una branch coverage nulla in quanto il metodo non contiene nessun branch [\[Immagine 22\]](#).

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [\[Immagine 23\]](#) fornito da **Jacoco** per il metodo `createDatabase()`, si è visto che non viene coperto un blocco catch.

Il blocco catch a riga **65** può essere coperto sollevando l'eccezione `ServiceException`. Per implementare il caso di test descritto sarebbe necessario uno studio approfondito del codice; quindi, si è deciso di non riservare ulteriore effort per il blocco considerato.

Analizzando il metodo si è notato che una delle eccezioni che può lanciare è `DuplicateDatabaseException`. Dal nome dell'eccezione si può capire facilmente che questa viene lanciata se si tenta di creare un database che ha lo stesso nome di uno già esistente. La classe di test è stata quindi migliorata attraverso il parametro booleano `duplicate`, che specifica se eseguire o meno il test che lancia l'eccezione descritta.

Il report sulla coverage dopo i miglioramenti apportati alla classe di test non mostra nessun miglioramento (come atteso): lo scopo dell'introduzione dei miglioramenti non era quello di incrementare il valore associato alle metriche di coverage, ma quello di provare a sollevare l'eccezione lanciabile dal metodo.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 24\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **76%** ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni.

4.1.2. public void dropDatabase(String databaseName)

Questo metodo permette di eliminare il database il cui nome viene specificato in input.

La partizione in classi di equivalenza definita è stata la seguente:

- `databaseName: {valid_string, empty_string}`

Per testare il funzionamento del metodo, in fase di configurazione viene creato un database, tramite il metodo `createDatabase()`, con `databaseName` uguale a "test_database". Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{databaseName}`
 - `{"test_database"}` → `no exception`
 - `{"database"}` → `no database found exception`
 - `{empty_string}` → `illegal argument exception`

Il caso di test `{empty_string}` non ha rispettato il comportamento atteso. Infatti, non ha sollevato nessuna eccezione, quindi il parametro `databaseName` può essere anche una stringa vuota. A seguito di questa osservazione, è stato eliminato questo caso di test.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **81%** ed una branch coverage nulla in quanto il metodo non contiene nessun branch [\[Immagine 22\]](#).

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [\[Immagine 25\]](#) fornito da **Jacoco** per il metodo `dropDatabase()`, si è visto che non viene coperto un blocco catch.

Il blocco catch a riga **101** può essere coperto sollevando l'eccezione `ServiceException`. Come in precedenza, si è deciso di non riservare effort per il blocco considerato.

Analizzando il metodo si è notato che le eccezioni che può lanciare sono:

- `UndefinedDatabaseException`: il database non è presente nel sistema.
- `CannotDropCurrentDatabaseException`: si cerca di eliminare il database attualmente selezionato.

La prima eccezione viene lanciata dal caso di test `{"database"}`.

Per la terza eccezione è stato inserito il parametro booleano `dropCurrent` all'interno della classe di test, che specifica se eseguire o meno il test che lancia l'eccezione descritta. È stato definito quindi il seguente caso di test:

- `{"test_database"}` → `cannot drop current database exception`

Il test non ha rispettato il comportamento atteso. Infatti, ha sollevato l'eccezione `TajoInternalError`. Analizzando il codice della classe `ErrorMessages` si è scoperta la presenza di un **bug** relativo all'eccezione `CannotDropCurrentDatabaseException`: il messaggio relativo a questa eccezione non contiene nessun argomento, ma viene aggiunta una entry con numero di argomenti pari ad **1** [\[Immagine 36\]](#). Siccome il bug trovato non è collegato univocamente con il metodo sottoposto a test, è stato risolto inserendo **0** come numero di argomenti presenti nella stringa.

Anche in questo caso, il report sulla coverage dopo le modifiche apportate alla classe di test non mostra nessun miglioramento.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 26\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **81%** ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni.

4.1.3. `public TableDesc createExternalTable(String tableName, Schema schema, URI path, TableMeta meta, PartitionMethodDesc partitionMethodDesc)`

Questo metodo permette di creare una nuova tabella le cui caratteristiche vengono specificate in input.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome.

Il parametro:

- `tableName` è una stringa che indica il nome della tabella da creare.
- `schema` rappresenta la struttura della tabella da creare.
- `path` indica il percorso della directory fisica associato alla tabella.
- `meta` contiene tutte le informazioni per analizzare la tabella frammentata.
- `partitionMethodDesc` contiene le informazioni relative ai tipi di partizioni della tabella.

Inizialmente, la partizione in classi di equivalenza definita è stata la seguente:

- `tableName`: {valid_string, empty_string}
- `schema`: {valid_schema, null}
- `path`: {valid_path, invalid_path}
- `meta`: {valid_meta, null}
- `partitionMethodDesc`: {valid_part, null}

Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {`tableName`, `schema`, `path`, `meta`, `partitionMethodDesc`}
 - {valid_string, valid_schema, valid_path, valid_meta, valid_part} → no exception
 - {valid_string, null, valid_path, valid_meta, null} → null pointer exception
 - {empty_string, valid_schema, valid_path, valid_meta, null} → no exception
 - {valid_string, valid_schema, invalid_path, valid_meta, valid_part} → no path exception

Il caso di test {valid_string, null, valid_path, valid_meta, null} non ha rispettato il comportamento atteso. Infatti, non ha sollevato nessuna eccezione, quindi i parametri `schema` e `partitionMethodDesc` possono essere istanze nulle. A seguito di questa osservazione, è stata rimossa l'eccezione prevista per il caso di test.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **92%** ed una branch coverage pari al **100%** [\[Immagine 22\]](#).

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [\[Immagine 27\]](#) fornito da **Jacoco** per il metodo `createExternalTable()`, si è visto che non viene coperto un blocco catch.

Il blocco catch a riga **168** può essere coperto sollevando l'eccezione `ServiceException`. Come in precedenza, si è deciso di non riservare effort per il blocco considerato.

Analizzando il metodo si è notato che le eccezioni che può lanciare sono:

- `UnavailableTableLocationException`: il path associato alla tabella non è presente nel sistema.

- `DuplicateTableException`: la tabella che si vuole inserire è già presente nel sistema.

La prima eccezione viene lanciata dal caso di test {valid_string, valid_schema, invalid_path, valid_meta, valid_part}.

Per la seconda eccezione è stato inserito il parametro booleano `duplicate` all'interno della classe di test, che specifica se eseguire o meno il test che lancia l'eccezione descritta.

Anche in questo caso, il report sulla coverage dopo le modifiche apportate alla classe di test non mostra nessun miglioramento.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 28\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

La test suite sviluppata è stata considerata adeguata, in quanto ha prodotto una statement coverage pari al **92%** ed una branch coverage pari al **100%**, ed è risultata estremamente robusta in quanto è riuscita a rilevare tutte le mutazioni.

4.2. SessionConnection

Questa classe implementa una connessione ad una sessione di **Tajo**. I metodi sperimentati per questa classe sono stati `updateSessionVariables()` e `getSessionVariable()` attraverso lo sviluppo di una test suite apposita.

4.2.1. `public Map<String, String> updateSessionVariables(Map<String, String> variables)`

Questo metodo permette di aggiornare le variabili di una sessione attraverso il parametro `variables`.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. Il parametro `variables` è una `HashMap` che associa i nomi delle variabili al valore di queste. La partizione in classi di equivalenza definita è stata la seguente:

- `variables: {valid_map, invalid_map}`

Per lo sviluppo dei casi di test è stato utilizzato un approccio unidimensionale, considerando quindi i vari parametri in modo indipendente e cercando di coprire con almeno un test tutte le classi di equivalenza definite. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- {variables}
 - {valid_map} → no exception
 - {invalid_map} → illegal argument exception

Il caso di test {invalid_map} non ha rispettato il comportamento atteso. Infatti, non ha sollevato nessuna eccezione, quindi le stringhe del parametro `variables` possono contenere qualsiasi carattere ed essere stringhe vuote. A seguito di questa osservazione, è stato eliminato il caso di test considerato.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **88%** ed una branch coverage nulla in quanto il metodo non contiene nessun branch [\[Immagine 29\]](#).

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [\[Immagine 30\]](#) fornito da **Jacoco** per il metodo `updateSessionVariables()`, si è visto che non viene coperto un blocco catch.

Il blocco catch a riga **216** può essere coperto sollevando l'eccezione `ServiceException`. Come in precedenza, si è deciso di non riservare effort per il blocco considerato.

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 31\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

4.2.2. public String getSessionVariable(String varname)

Questo metodo permette di ottenere il valore associato alla variabile il cui nome viene specificato attraverso il parametro `varname`.

Applicando il metodo di **Domain Partitioning** sono stati individuati i possibili valori che possono essere assunti dai parametri, basandosi sulla semantica del nome. Il parametro `varname` è una stringa che indica la chiave associata ad una variabile. Inizialmente, la partizione in classi di equivalenza definita è stata la seguente:

- `varname: {valid_string, invalid_string}`

Per testare il funzionamento del metodo, in fase di configurazione viene inserita una variabile, tramite il metodo `updateSessionVariables()`, con chiave `"test_key"` e valore `"test_value"`. Per la scelta dei valori di ogni classe di equivalenza è stata applicata la **Boundary Values Analysis**, che ha portato allo sviluppo dei seguenti casi di test:

- `{varname}`
 - `{"test_key"}` → no exception
 - `{"key"}` → no variable exception
 - `{invalid_string}` → illegal argument exception

Il caso di test `{invalid_string}` non ha rispettato il comportamento atteso. Infatti, ha sollevato l'eccezione `NoSuchSessionVariableException`, quindi il parametro `varname` può contenere qualsiasi carattere ed essere una stringa vuota. A seguito di questa osservazione, è stato eliminato il caso di test considerato.

La test suite così definita ha permesso di raggiungere una statement coverage pari al **79%** ed una branch coverage pari al **75%** [\[Immagine 29\]](#).

Non riuscendo a migliorare la test suite attraverso un approccio **Black-Box**, si è passati a quello **White-Box**. Analizzando il report [\[Immagine 32\]](#) fornito da **Jacoco** per il metodo `getSessionVariable()`, si è visto che non viene coperto un blocco catch ed un branch.

Il blocco catch a riga **68** può essere coperto sollevando l'eccezione `ServiceException`. Come in precedenza, si è deciso di non riservare effort per il blocco considerato.

Il branch a riga **72** (`!isThisError(response.getState(), NO_SUCH_SESSION_VARIABLE)`) può essere coperto se la variabile di interesse è presente nel sistema ma non nella cache lato client. Per pulire la cache è stato necessario utilizzare il metodo `setField()` di **Mockito**, in quanto l'attributo `sessionVarsCache` di `SessionConnection` è privato e non è presente un metodo che permette di azzerarla. Il metodo `setField()` permette di accedere, attraverso *reflection*, ai vari campi e metodi di una istanza, anche se sono privati. È stato, quindi, inserito il parametro booleano `improvement` all'interno della classe di test, che specifica se eseguire o meno il test descritto.

La test suite così definita ha permesso di raggiungere il **90%** di statement coverage e il **100%** di branch coverage: risulta quindi essere una test suite adeguata per il metodo considerato [\[Immagine 33\]](#) [\[Immagine 34\]](#).

Al fine di migliorare l'adeguatezza della test suite, è stato eseguito **Mutation Testing** tramite il framework **Pit**. Dal report [\[Immagine 35\]](#) si può vedere che tutte le mutazioni sono state rilevate, non è stato quindi necessario definire ulteriori casi di test.

5. Immagini

Immagine 1 – Coverage JCS

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
get(Object)		100%		100%	0	2	0	2	0	1
put(Object, Object)		100%		n/a	0	1	0	2	0	1
remove(Object)		100%		n/a	0	1	0	2	0	1
CacheAccess(CompositeCache)		100%		n/a	0	1	0	3	0	1
getStats()		100%		n/a	0	1	0	1	0	1
dispose()		100%		n/a	0	1	0	2	0	1
static {...}		90%		50%	1	2	0	1	0	1
put(Object, Object, IElementAttributes)		60%		50%	2	3	4	11	0	1
clear()		45%		n/a	0	1	2	5	0	1
freeMemoryElements(int)		0%		n/a	1	1	8	8	1	1
resetElementAttributes(Object, IElementAttributes)		0%		0%	2	2	5	5	1	1
putSafe(Object, Object)		0%		0%	2	2	4	4	1	1
ensureCacheManager()		0%		0%	3	3	5	5	1	1
getElementAttributes(Object)		0%		n/a	1	1	6	6	1	1
destroy()		0%		n/a	1	1	5	5	1	1
defineRegion(String, ICompositeCacheAttributes, IElementAttributes)		0%		n/a	1	1	2	2	1	1
defineRegion(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
getAccess(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
defineRegion(String)		0%		n/a	1	1	2	2	1	1
getAccess(String)		0%		n/a	1	1	2	2	1	1
destroy(Object)		0%		n/a	1	1	2	2	1	1
getCacheElement(Object)		0%		n/a	1	1	1	1	1	1
resetElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
setDefaultElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
setCacheAttributes(ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
getElementAttributes()		0%		n/a	1	1	1	1	1	1
getDefaultElementAttributes()		0%		n/a	1	1	1	1	1	1
getStatistics()		0%		n/a	1	1	1	1	1	1
getCacheAttributes()		0%		n/a	1	1	1	1	1	1
remove()		0%		n/a	1	1	2	2	1	1
Total	246 of 327	24%	11 of 16	31%	28	38	64	87	21	30

Immagine 2 – Coverage WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
get(long, long)		100%		100%	0	2	0	10	0	1
clear()		100%		n/a	0	1	0	7	0	1
close()		100%		100%	0	2	0	3	0	1
getLastEntry(long)		100%		100%	0	2	0	4	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
WriteCache(ByteBufAllocator, long)		100%		n/a	0	1	0	2	0	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		96%		62%	3	5	3	20	0	1
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda\$forEach\$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
count()		0%		n/a	1	1	1	1	1	1
Total	249 of 617	59%	23 of 38	39%	20	35	48	120	6	16

Immagine 3 – Coverage put

```

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

```

Immagine 4 – Coverage WriteCache a seguito del miglioramento della test suite

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
get(long, long)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 10	0 1
clear()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 7	0 1
close()	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 3	0 1
getLastEntry(long)	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 4	0 1
alignToPowerOfTwo(long)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
static {...}	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
align64(int)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
WriteCache(ByteBufAllocator, long)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
size()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
count()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
put(long, long, ByteBuf)	<div><div></div></div>	98%	<div><div></div></div>	87%	1 5	1 20	0 1
WriteCache(ByteBufAllocator, long, int)	<div><div></div></div>	98%	<div><div></div></div>	66%	2 4	0 25	0 1
forEach(WriteCache.EntryConsumer)	<div><div></div></div>	0%	<div><div></div></div>	0%	8 8	32 32	1 1
lambda\$forEach\$0(long, long, long, long)	<div><div></div></div>	0%	<div><div></div></div>	0%	2 2	8 8	1 1
isEmpty()	<div><div></div></div>	0%	<div><div></div></div>	0%	2 2	1 1	1 1
deleteLedger(long)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	2 2	1 1
Total	239 of 617	61%	21 of 38	44%	16 35	44 120	4 16

Immagine 5 – Coverage put a seguito del miglioramento della test suite

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

Immagine 6 – Put report put

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

Immagine 7 – Coverage get

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

Immagine 8 – Pit report get

```

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

```

Immagine 9 – Coverage getLastEntry

```

199.     public ByteBuf getLastEntry(long ledgerId) {
200.         long lastEntryId = lastEntryMap.get(ledgerId);
201.         if (lastEntryId == -1) {
202.             // Ledger not found in write cache
203.             return null;
204.         } else {
205.             return get(ledgerId, lastEntryId);
206.         }
207.     }

```

Immagine 10 – Pit report getLastEntry

```

199     public ByteBuf getLastEntry(long ledgerId) {
200         long lastEntryId = lastEntryMap.get(ledgerId);
201         if (lastEntryId == -1) {
202             // Ledger not found in write cache
203             return null;
204         } else {
205             return get(ledgerId, lastEntryId);
206         }
207     }

```

Immagine 11 – Coverage BookieImpl

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed Cxty	Missed Lines	Missed Methods
readJournal()		100%		100%	0 2	0 8	0 1
getCurrentDirectories(File[])		100%		100%	0 2	0 4	0 1
initializeStateManager()		100%		n/a	0 1	0 1	0 1
fenceLedger(long, byte[])		100%		n/a	0 1	0 2	0 1
getLedgerForEntry(ByteBuf, byte[])		100%		n/a	0 1	0 2	0 1
BookieImpl(ServerConfiguration)		100%		n/a	0 1	0 2	0 1
buildLedgerStorage(ServerConfiguration)		100%		n/a	0 1	0 3	0 1
getJournal(long)		100%		n/a	0 1	0 1	0 1
createLedgerDirsManager(ServerConfiguration, DiskChecker, StatsLogger)		100%		n/a	0 1	0 1	0 1
readLastAddConfirmed(long)		100%		n/a	0 1	0 2	0 1
createDiskChecker(ServerConfiguration)		100%		n/a	0 1	0 1	0 1
getCurrentDirectory(File)		100%		n/a	0 1	0 1	0 1
getLedgerDirsListener()		100%		n/a	0 1	0 1	0 1
isRunning()		100%		n/a	0 1	0 1	0 1
shutdown()		100%		n/a	0 1	0 1	0 1
static {...}		100%		n/a	0 1	0 1	0 1
readEntry(long, long)		89%		75%	1 3	1 16	0 1
recoveryAddEntry(ByteBuf, BookkeeperInternalCallbacks.WriteCallback, Object, byte[])		86%		100%	0 2	3 20	0 1
BookieImpl(ServerConfiguration, StatsLogger, ByteBufAllocator, Supplier)		84%		64%	5 8	10 58	0 1
addEntryInternal(LedgerDescriptor, ByteBuf, boolean, BookkeeperInternalCallbacks.WriteCallback, Object, byte[])		84%		50%	4 5	3 20	0 1
addEntry(ByteBuf, boolean, BookkeeperInternalCallbacks.WriteCallback, Object, byte[])		83%		75%	1 3	5 23	0 1

Immagine 12 – Coverage addEntry

```

1400.     public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)
1401.     throws IOException, BookieException, InterruptedException {
1402.         long requestNanos = MathUtils.nowInNano();
1403.         boolean success = false;
1404.         int entrySize = 0;
1405.         try {
1406.             LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
1407.             synchronized (handle) {
1408.                 if (handle.isFenced()) {
1409.                     throw BookieException
1410.                         .create(BookieException.Code.LedgerFencedException);
1411.                 }
1412.                 entrySize = entry.readableBytes();
1413.                 addEntryInternal(handle, entry, ackBeforeSync, cb, ctx, masterKey);
1414.             }
1415.             success = true;
1416.         } catch (NoWritableLedgerDirException e) {
1417.             stateManager.transitionToReadOnlyMode();
1418.             throw new IOException(e);
1419.         } finally {
1420.             long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1421.             if (success) {
1422.                 bookieStats.getAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1423.                 bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
1424.             } else {
1425.                 bookieStats.getAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1426.                 bookieStats.getAddBytesStats().registerFailedValue(entrySize);
1427.             }
1428.
1429.             entry.release();
1430.         }
1431.     }

```

Immagine 13 – Coverage BookieImpl a seguito del miglioramento della test suite

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
readJournal()		100%		100%	0	2	0	8	0	1
getCurrentDirectories(File[])		100%		100%	0	2	0	4	0	1
initializeStateManager()		100%		n/a	0	1	0	1	0	1
fenceLedger(long, byte[])		100%		n/a	0	1	0	2	0	1
getLedgerForEntry(ByteBuf, byte[])		100%		n/a	0	1	0	2	0	1
BookieImpl(ServerConfiguration)		100%		n/a	0	1	0	2	0	1
buildLedgerStorage(ServerConfiguration)		100%		n/a	0	1	0	3	0	1
getJournal(long)		100%		n/a	0	1	0	1	0	1
createLedgerDirsManager(ServerConfiguration, DiskChecker, StatsLogger)		100%		n/a	0	1	0	1	0	1
readLastAddConfirmed(long)		100%		n/a	0	1	0	2	0	1
createDiskChecker(ServerConfiguration)		100%		n/a	0	1	0	1	0	1
getCurrentDirectory(File)		100%		n/a	0	1	0	1	0	1
getLedgerDirsListener()		100%		n/a	0	1	0	1	0	1
isRunning()		100%		n/a	0	1	0	1	0	1
shutdown()		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	1	0	1
readEntry(long, long)		89%		75%	1	3	1	16	0	1
addEntry(ByteBuf, boolean, BookkeeperInternalCallbacks, WriteCallback, Object, byte[])		87%		100%	0	3	3	23	0	1
recoveryAddEntry(ByteBuf, BookkeeperInternalCallbacks, WriteCallback, Object, byte[])		86%		100%	0	2	3	20	0	1
BookieImpl(ServerConfiguration, StatsLogger, ByteBufAllocator, Supplier)		84%		64%	5	8	10	58	0	1

Immagine 14 – Coverage addEntry a seguito del miglioramento della test suite

```

1400.     public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)
1401.     throws IOException, BookieException, InterruptedException {
1402.         long requestNanos = MathUtils.nowInNano();
1403.         boolean success = false;
1404.         int entrySize = 0;
1405.         try {
1406.             LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
1407.             synchronized (handle) {
1408.                 if (handle.isFenced()) {
1409.                     throw BookieException
1410.                         .create(BookieException.Code.LedgerFencedException);
1411.                 }
1412.                 entrySize = entry.readableBytes();
1413.                 addEntryInternal(handle, entry, ackBeforeSync, cb, ctx, masterKey);
1414.             }
1415.             success = true;
1416.         } catch (NoWritableLedgerDirException e) {
1417.             stateManager.transitionToReadOnlyMode();
1418.             throw new IOException(e);
1419.         } finally {
1420.             long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1421.             if (success) {
1422.                 bookieStats.getAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1423.                 bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
1424.             } else {
1425.                 bookieStats.getAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1426.                 bookieStats.getAddBytesStats().registerFailedValue(entrySize);
1427.             }
1428.             entry.release();
1429.         }
1430.     }
1431. }

```

Immagine 15 – Pit report addEntry

```

1400     public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)
1401     throws IOException, BookieException, InterruptedException {
1402         long requestNanos = MathUtils.nowInNano();
1403         boolean success = false;
1404         int entrySize = 0;
1405         try {
1406             LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
1407             synchronized (handle) {
1408. 1         if (handle.isFenced()) {
1409                 throw BookieException
1410                     .create(BookieException.Code.LedgerFencedException);
1411             }
1412             entrySize = entry.readableBytes();
1413. 1         addEntryInternal(handle, entry, ackBeforeSync, cb, ctx, masterKey);
1414             }
1415             success = true;
1416         } catch (NoWritableLedgerDirException e) {
1417             stateManager.transitionToReadOnlyMode();
1418             throw new IOException(e);
1419         } finally {
1420             long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1421. 1         if (success) {
1422. 1             bookieStats.getAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1423. 1             bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
1424         } else {
1425. 1             bookieStats.getAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1426. 1             bookieStats.getAddBytesStats().registerFailedValue(entrySize);
1427         }
1428             entry.release();
1429         }
1430     }
1431 }

```

Immagine 16 – Coverage recoveryAddEntry

```
1317.     public void recoveryAddEntry(ByteBuf entry, WriteCallback cb, Object ctx, byte[] masterKey)
1318.     throws IOException, BookieException, InterruptedException {
1319.         long requestNanos = MathUtils.nowInNano();
1320.         boolean success = false;
1321.         int entrySize = 0;
1322.         try {
1323.             LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
1324.             synchronized (handle) {
1325.                 entrySize = entry.readableBytes();
1326.                 addEntryInternal(handle, entry, false /* ackBeforeSync */, cb, ctx, masterKey);
1327.             }
1328.             success = true;
1329.         } catch (NoWritableLedgerDirException e) {
1330.             stateManager.transitionToReadOnlyMode();
1331.             throw new IOException(e);
1332.         } finally {
1333.             long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1334.             if (success) {
1335.                 bookieStats.getRecoveryAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1336.                 bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
1337.             } else {
1338.                 bookieStats.getRecoveryAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1339.                 bookieStats.getAddBytesStats().registerFailedValue(entrySize);
1340.             }
1341.             entry.release();
1342.         }
1343.     }
1344. }
```

Immagine 17 – Pit report recoveryAddEntry

```
1317     public void recoveryAddEntry(ByteBuf entry, WriteCallback cb, Object ctx, byte[] masterKey)
1318     throws IOException, BookieException, InterruptedException {
1319         long requestNanos = MathUtils.nowInNano();
1320         boolean success = false;
1321         int entrySize = 0;
1322         try {
1323             LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
1324             synchronized (handle) {
1325                 entrySize = entry.readableBytes();
1326                 addEntryInternal(handle, entry, false /* ackBeforeSync */, cb, ctx, masterKey);
1327             }
1328             success = true;
1329         } catch (NoWritableLedgerDirException e) {
1330             stateManager.transitionToReadOnlyMode();
1331             throw new IOException(e);
1332         } finally {
1333             long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1334             if (success) {
1335                 bookieStats.getRecoveryAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1336                 bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
1337             } else {
1338                 bookieStats.getRecoveryAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1339                 bookieStats.getAddBytesStats().registerFailedValue(entrySize);
1340             }
1341             entry.release();
1342         }
1343     }
1344. }
```

Immagine 18 – Coverage readEntry

```
1447. public ByteBuf readEntry(long ledgerId, long entryId)
1448.     throws IOException, NoLedgerException {
1449.     long requestNanos = MathUtils.nowInNano();
1450.     boolean success = false;
1451.     int entrySize = 0;
1452.     try {
1453.         LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1454.         if (LOG.isTraceEnabled()) {
1455.             LOG.trace("Reading {}@{}", entryId, ledgerId);
1456.         }
1457.         ByteBuf entry = handle.readEntry(entryId);
1458.         bookieStats.getReadBytes().add(entry.readableBytes());
1459.         success = true;
1460.         return entry;
1461.     } finally {
1462.         long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1463.         if (success) {
1464.             bookieStats.getReadEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1465.             bookieStats.getReadBytesStats().registerSuccessfulValue(entrySize);
1466.         } else {
1467.             bookieStats.getReadEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1468.             bookieStats.getReadEntryStats().registerFailedValue(entrySize);
1469.         }
1470.     }
1471. }
```

Immagine 19 – Pit report readEntry

```
1447 public ByteBuf readEntry(long ledgerId, long entryId)
1448     throws IOException, NoLedgerException {
1449     long requestNanos = MathUtils.nowInNano();
1450     boolean success = false;
1451     int entrySize = 0;
1452     try {
1453         LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1454         if (LOG.isTraceEnabled()) {
1455             LOG.trace("Reading {}@{}", entryId, ledgerId);
1456         }
1457         ByteBuf entry = handle.readEntry(entryId);
1458         bookieStats.getReadBytes().add(entry.readableBytes());
1459         success = true;
1460         return entry;
1461     } finally {
1462         long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1463         if (success) {
1464             bookieStats.getReadEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1465             bookieStats.getReadBytesStats().registerSuccessfulValue(entrySize);
1466         } else {
1467             bookieStats.getReadEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1468             bookieStats.getReadEntryStats().registerFailedValue(entrySize);
1469         }
1470     }
1471 }
```

Immagine 20 – Coverage readLastAddConfirmed

```
1473. public long readLastAddConfirmed(long ledgerId) throws IOException {
1474.     LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1475.     return handle.getLastAddConfirmed();
1476. }
```

Immagine 21 – Pit report readLastAddConfirmed

```
1473 public long readLastAddConfirmed(long ledgerId) throws IOException {
1474     LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1475     return handle.getLastAddConfirmed();
1476 }
```


Immagine 22 – Coverage CatalogAdminClientImpl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• createExternalTable(String, Schema, URI, TableMeta)		100%		n/a	0	1	0	1	0	1
• CatalogAdminClientImpl(SessionConnection)		100%		n/a	0	1	0	3	0	1
• createExternalTable(String, Schema, URI, TableMeta, PartitionMethodDesc)		92%		100%	0	3	2	21	0	1
• dropTable(String, boolean)		83%		n/a	0	1	2	14	0	1
• dropDatabase(String)		81%		n/a	0	1	2	10	0	1
• existTable(String)		80%		100%	0	2	2	9	0	1
• existDatabase(String)		79%		100%	0	2	2	8	0	1
• createDatabase(String)		76%		n/a	0	1	2	8	0	1
• getTableList(String)		76%		n/a	0	1	2	7	0	1
• getAllDatabaseNames()		66%		n/a	0	1	2	4	0	1
• getIndex(String, String[])		0%		0%	2	2	12	12	1	1
• existIndex(String, String[])		0%		0%	2	2	9	9	1	1
• getTableDesc(String)		0%		0%	2	2	9	9	1	1
• getPartitionsOfTable(String)		0%		n/a	1	1	11	11	1	1
• getFunctions(String)		0%		0%	2	2	8	8	1	1
• getIndex(String)		0%		n/a	1	1	7	7	1	1
• getIndexes(String)		0%		n/a	1	1	8	8	1	1
• existIndex(String)		0%		n/a	1	1	4	4	1	1
• hasIndexes(String)		0%		n/a	1	1	4	4	1	1
• dropIndex(String)		0%		n/a	1	1	4	4	1	1
• dropTable(String)		0%		n/a	1	1	2	2	1	1
• close()		0%		n/a	1	1	1	1	1	1
Total	378 of 625	39%	8 of 16	50%	16	30	95	164	12	22

Immagine 23 – Coverage createDatabase

```

53.  @Override
54.  public void createDatabase(final String databaseName) throws DuplicateDatabaseException {
55.
56.
57.
58.      try {
59.          final BlockingInterface stub = conn.getTMStub();
60.          final ReturnState state = stub.createDatabase(null, conn.getSessionedString(databaseName));
61.
62.          throwsIfThisError(state, DuplicateDatabaseException.class);
63.          ensureOk(state);
64.
65.      } catch (ServiceException e) {
66.          throw new RuntimeException(e);
67.      }
68.  }

```

Immagine 24 – Pit report createDatabase

```

53  @Override
54  public void createDatabase(final String databaseName) throws DuplicateDatabaseException {
55
56
57
58      try {
59          final BlockingInterface stub = conn.getTMStub();
60          final ReturnState state = stub.createDatabase(null, conn.getSessionedString(databaseName));
61
62  1  throwsIfThisError(state, DuplicateDatabaseException.class);
63      ensureOk(state);
64
65  } catch (ServiceException e) {
66      throw new RuntimeException(e);
67  }
68  }

```

Immagine 25 – Coverage dropDatabase

```
88.  @Override
89.  public void dropDatabase(final String databaseName)
90.      throws UndefinedDatabaseException, InsufficientPrivilegeException, CannotDropCurrentDatabaseException {
91.
92.      try {
93.          final BlockingInterface stub = conn.getTMStub();
94.          final ReturnState state = stub.dropDatabase(null, conn.getSessionedString(databaseName));
95.
96.          throwsIfThisError(state, UndefinedDatabaseException.class);
97.          throwsIfThisError(state, InsufficientPrivilegeException.class);
98.          throwsIfThisError(state, CannotDropCurrentDatabaseException.class);
99.          ensureOk(state);
100.
101.      } catch (ServiceException e) {
102.          throw new RuntimeException(e);
103.      }
104.  }
```

Immagine 26 – Pit report dropDatabase

```
88  @Override
89  public void dropDatabase(final String databaseName)
90      throws UndefinedDatabaseException, InsufficientPrivilegeException, CannotDropCurrentDatabaseException {
91
92      try {
93          final BlockingInterface stub = conn.getTMStub();
94          final ReturnState state = stub.dropDatabase(null, conn.getSessionedString(databaseName));
95
96  1  throwsIfThisError(state, UndefinedDatabaseException.class);
97  1  throwsIfThisError(state, InsufficientPrivilegeException.class);
98  1  throwsIfThisError(state, CannotDropCurrentDatabaseException.class);
99      ensureOk(state);
100
101  } catch (ServiceException e) {
102      throw new RuntimeException(e);
103  }
104  }
```

Immagine 27 – Coverage createExternalTable

```
137.  @Override
138.  public TableDesc createExternalTable(String tableName, @Nullable Schema schema, URI path, TableMeta meta)
139.      throws DuplicateTableException, UnavailableTableLocationException, InsufficientPrivilegeException {
140.      return createExternalTable(tableName, schema, path, meta, null);
141.  }
142.
143.  @Override
144.  public TableDesc createExternalTable(final String tableName, @Nullable final Schema schema, final URI path,
145.                                     final TableMeta meta, final PartitionMethodDesc partitionMethodDesc)
146.      throws DuplicateTableException, InsufficientPrivilegeException, UnavailableTableLocationException {
147.
148.      final NettyClientBase client = conn.getTajoMasterConnection();
149.      conn.checkSessionAndGet(client);
150.      final BlockingInterface tajoMasterService = client.getStub();
151.
152.      final ClientProtos.CreateTableRequest.Builder builder = ClientProtos.CreateTableRequest.newBuilder();
153.      builder.setSessionId(conn.sessionId);
154.      builder.setName(tableName);
155.      ◆ if (schema != null) {
156.          builder.setSchema(schema.getProto());
157.      }
158.      builder.setMeta(meta.getProto());
159.      builder.setPath(path.toString());
160.
161.      ◆ if (partitionMethodDesc != null) {
162.          builder.setPartition(partitionMethodDesc.getProto());
163.      }
164.
165.      TableResponse res;
166.      try {
167.          res = tajoMasterService.createExternalTable(null, builder.build());
168.      } catch (ServiceException e) {
169.          throw new RuntimeException(e);
170.      }
171.
172.      throwsIfThisError(res.getState(), DuplicateTableException.class);
173.      throwsIfThisError(res.getState(), InsufficientPrivilegeException.class);
174.      throwsIfThisError(res.getState(), UnavailableTableLocationException.class);
175.
176.      ensureOk(res.getState());
177.      return new TableDesc(res.getTable());
178.  }
```

Immagine 28 – Pit report createExternalTable

```

143 @Override
144 public TableDesc createExternalTable(final String tableName, @Nullable final Schema schema, final URI path,
145                                     final TableMeta meta, final PartitionMethodDesc partitionMethodDesc)
146     throws DuplicateTableException, InsufficientPrivilegeException, UnavailableTableLocationException {
147
148     final NettyClientBase client = conn.getTajoMasterConnection();
149     conn.checkSessionAndGet(client);
150     final BlockingInterface tajoMasterService = client.getStub();
151
152     final ClientProtos.CreateTableRequest.Builder builder = ClientProtos.CreateTableRequest.newBuilder();
153     builder.setSessionId(conn.sessionId);
154     builder.setName(tableName);
155     if (schema != null) {
156         builder.setSchema(schema.getProto());
157     }
158     builder.setMeta(meta.getProto());
159     builder.setPath(path.toString());
160
161     if (partitionMethodDesc != null) {
162         builder.setPartition(partitionMethodDesc.getProto());
163     }
164
165     TableResponse res;
166     try {
167         res = tajoMasterService.createExternalTable(null, builder.build());
168     } catch (ServiceException e) {
169         throw new RuntimeException(e);
170     }
171
172     throwsIfThisError(res.getState(), DuplicateTableException.class);
173     throwsIfThisError(res.getState(), InsufficientPrivilegeException.class);
174     throwsIfThisError(res.getState(), UnavailableTableLocationException.class);
175
176     ensureOk(res.getState());
177     return new TableDesc(res.getTable());
178 }

```

Immagine 29 – Coverage SessionConnection

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
static {...}		100%		n/a	0	1	0	2	0	1
updateSessionVarsCache(Map)		100%		n/a	0	1	0	5	0	1
getSessionedString(String)		100%		100%	0	2	0	5	0	1
getTMSStub()		100%		n/a	0	1	0	4	0	1
getTajoMasterAddr()		100%		n/a	0	1	0	1	0	1
getProperties()		100%		n/a	0	1	0	1	0	1
close()		94%		50%	1	2	2	10	0	1
SessionConnection(ServiceTracker, String, KeyValueType)		88%		n/a	0	1	3	17	0	1
updateSessionVariables(Map)		88%		n/a	0	1	2	15	0	1
unsetSessionVariables(List)		83%		n/a	0	1	2	12	0	1
getSessionVariable(String)		79%		75%	1	3	4	15	0	1
getTajoMasterConnection()		79%		75%	1	3	2	8	0	1
existSessionVariable(String)		78%		100%	0	2	2	9	0	1
selectDatabase(String)		77%		n/a	0	1	2	9	0	1
checkSessionAndGet(NettyClientBase)		55%		62%	3	5	5	18	0	1
reconnect()		0%		0%	6	6	30	30	1	1
getAllSessionVariables()		0%		n/a	1	1	9	9	1	1
getCurrentDatabase()		0%		n/a	1	1	9	9	1	1
isConnected()		0%		0%	2	2	5	5	1	1
getClientSideSessionVars()		0%		n/a	1	1	1	1	1	1
setSessionId(TajoProtos.SessionIdProto)		0%		n/a	1	1	2	2	1	1
getSessionId()		0%		n/a	1	1	1	1	1	1
getBaseDatabase()		0%		n/a	1	1	1	1	1	1
getUserInfo()		0%		n/a	1	1	1	1	1	1
Total	327 of 730	55%	18 of 34	47%	21	41	83	190	9	24

Immagine 30 – Coverage updateSessionVariables

```
201. public Map<String, String> updateSessionVariables(final Map<String, String> variables) {
202.     NettyClientBase client = getTajoMasterConnection();
203.     checkSessionAndGet(client);
204.
205.     BlockingInterface tajoMasterService = client.getStub();
206.     KeyValueSet keyValueSet = new KeyValueSet();
207.     keyValueSet.putAll(variables);
208.     UpdateSessionVariableRequest request = UpdateSessionVariableRequest.newBuilder()
209.         .setSessionId(sessionId)
210.         .setSessionVars(keyValueSet.getProto()).build();
211.
212.     SessionUpdateResponse response;
213.
214.     try {
215.         response = tajoMasterService.updateSessionVariables(null, request);
216.     } catch (ServiceException e) {
217.         throw new RuntimeException(e);
218.     }
219.
220.     ensureOk(response.getState());
221.     updateSessionVarsCache(ProtoUtil.convertToMap(response.getSessionVars()));
222.     return Collections.unmodifiableMap(sessionVarsCache);
223. }
```

Immagine 31 – Pit report updateSessionVariables

```
201 public Map<String, String> updateSessionVariables(final Map<String, String> variables) {
202     NettyClientBase client = getTajoMasterConnection();
203 1 checkSessionAndGet(client);
204
205     BlockingInterface tajoMasterService = client.getStub();
206     KeyValueSet keyValueSet = new KeyValueSet();
207 1 keyValueSet.putAll(variables);
208     UpdateSessionVariableRequest request = UpdateSessionVariableRequest.newBuilder()
209         .setSessionId(sessionId)
210         .setSessionVars(keyValueSet.getProto()).build();
211
212     SessionUpdateResponse response;
213
214     try {
215         response = tajoMasterService.updateSessionVariables(null, request);
216     } catch (ServiceException e) {
217         throw new RuntimeException(e);
218     }
219
220     ensureOk(response.getState());
221 1 updateSessionVarsCache(ProtoUtil.convertToMap(response.getSessionVars()));
222 1 return Collections.unmodifiableMap(sessionVarsCache);
223 }
```

Immagine 32 – Coverage getSessionVariable

```

252. public String getSessionVariable(final String varname) throws NoSuchSessionVariableException {
253.     synchronized (sessionVarsCache) {
254.         // If a desired variable is client side one and exists in the cache, immediately return the variable.
255.         if (sessionVarsCache.containsKey(varname)) {
256.             return sessionVarsCache.get(varname);
257.         }
258.     }
259.
260.     NettyClientBase client = getTajoMasterConnection();
261.     checkSessionAndGet(client);
262.
263.     BlockingInterface stub = client.getStub();
264.     StringResponse response;
265.     try {
266.         response = stub.getSessionVariable(null, getSessionString(varname));
267.     } catch (ServiceException e) {
268.         throw new RuntimeException(e);
269.     }
270.
271.     if (isThisError(response.getState(), NO_SUCH_SESSION_VARIABLE)) {
272.         throw new NoSuchSessionVariableException(response.getState());
273.     }
274.
275.     ensureOk(response.getState());
276.     return response.getValue();
277. }
278.

```

Immagine 33 – Coverage SessionConnection a seguito del miglioramento della test suite

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
static {...}		100%		n/a	0	1	0	2	0	1
updateSessionVarsCache(Map)		100%		n/a	0	1	0	5	0	1
getSessionString(String)		100%		100%	0	2	0	5	0	1
getTMStub()		100%		n/a	0	1	0	4	0	1
getTajoMasterAddr()		100%		n/a	0	1	0	1	0	1
getProperties()		100%		n/a	0	1	0	1	0	1
close()		94%		50%	1	2	2	10	0	1
getSessionVariable(String)		90%		100%	0	3	2	15	0	1
SessionConnection(ServiceTracker, String, KeyValueSet)		88%		n/a	0	1	3	17	0	1
updateSessionVariables(Map)		88%		n/a	0	1	2	15	0	1
unsetSessionVariables(List)		83%		n/a	0	1	2	12	0	1
getTajoMasterConnection()		79%		75%	1	3	2	8	0	1
existSessionVariable(String)		78%		100%	0	2	2	9	0	1
selectDatabase(String)		77%		n/a	0	1	2	9	0	1
checkSessionAndGet(NettyClientBase)		55%		62%	3	5	5	18	0	1
reconnect()		0%		0%	6	6	30	30	1	1
getAllSessionVariables()		0%		n/a	1	1	9	9	1	1
getCurrentDatabase()		0%		n/a	1	1	9	9	1	1
isConnected()		0%		0%	2	2	5	5	1	1
getClientSideSessionVars()		0%		n/a	1	1	1	1	1	1
setSessionId(TajoIdProto.SessionIdProto)		0%		n/a	1	1	2	2	1	1
getSessionId()		0%		n/a	1	1	1	1	1	1
getBaseDatabase()		0%		n/a	1	1	1	1	1	1
getUserInfo()		0%		n/a	1	1	1	1	1	1
Total	320 of 730	56%	17 of 34	50%	20	41	81	190	9	24

Immagine 34 – Coverage getSessionVariable a seguito del miglioramento della test suite

```
252. public String getSessionVariable(final String varname) throws NoSuchSessionVariableException {
253.     synchronized (sessionVarsCache) {
254.         // If a desired variable is client side one and exists in the cache, immediately return the variable.
255.         if (sessionVarsCache.containsKey(varname)) {
256.             return sessionVarsCache.get(varname);
257.         }
258.     }
259.
260.     NettyClientBase client = getTajoMasterConnection();
261.     checkSessionAndGet(client);
262.
263.     BlockingInterface stub = client.getStub();
264.     StringResponse response;
265.     try {
266.         response = stub.getSessionVariable(null, getSessionedString(varname));
267.     } catch (ServiceException e) {
268.         throw new RuntimeException(e);
269.     }
270.
271.
272.     if (isThisError(response.getState(), NO_SUCH_SESSION_VARIABLE)) {
273.         throw new NoSuchSessionVariableException(response.getState());
274.     }
275.
276.     ensureOk(response.getState());
277.     return response.getValue();
278. }
```

Immagine 35 – Pit report getSessionVariable

```
252 public String getSessionVariable(final String varname) throws NoSuchSessionVariableException {
253     synchronized (sessionVarsCache) {
254         // If a desired variable is client side one and exists in the cache, immediately return the variable.
255 1   if (sessionVarsCache.containsKey(varname)) {
256 1   return sessionVarsCache.get(varname);
257     }
258 }
259
260 NettyClientBase client = getTajoMasterConnection();
261 1 checkSessionAndGet(client);
262
263 BlockingInterface stub = client.getStub();
264 StringResponse response;
265 try {
266     response = stub.getSessionVariable(null, getSessionedString(varname));
267 } catch (ServiceException e) {
268     throw new RuntimeException(e);
269 }
270
271
272 1 if (isThisError(response.getState(), NO_SUCH_SESSION_VARIABLE)) {
273     throw new NoSuchSessionVariableException(response.getState());
274 }
275
276 ensureOk(response.getState());
277 1 return response.getValue();
278 }
```

Immagine 36 – Tajo bug

```
// Syntax Error or Access Rule Violation
ADD_MESSAGE(SYNTAX_ERROR, "%s", 1);
ADD_MESSAGE(INSUFFICIENT_PRIVILEGE, "Insufficient privilege to %s", 1);
ADD_MESSAGE(CANNOT_DROP_CURRENT_DATABASE, "Cannot drop the current database", 1);
```

6. Riferimenti

6.1. JCS

- GitHub: <https://github.com/gabrielequatrana/JCSTest>
- Travis CI: <https://app.travis-ci.com/github/gabrielequatrana/JCSTest>

6.2. Bookkeeper

- GitHub: <https://github.com/gabrielequatrana/bookkeeper>
- Travis CI: <https://app.travis-ci.com/github/gabrielequatrana/bookkeeper>
- SonarCloud: https://sonarcloud.io/dashboard?id=gabrielequatrana_bookkeeper

6.3. Tajo

- GitHub: <https://github.com/gabrielequatrana/tajo>
- Travis CI: <https://app.travis-ci.com/github/gabrielequatrana/tajo>
- SonarCloud: https://sonarcloud.io/dashboard?id=gabrielequatrana_tajo