Ingegneria del Software

Software Testing and Mutation Coverage

Matteo Esposito
0285612

Introduzione

Il seguente report è stato stilato su due progetti Opensource mantenuti *dall'Apache Software Foundation* scelti secondo l'algoritmo presentato nel corso nello specifico il progetto comune al corso **BookKeeper** ed il progeto personale **Tajo.** La scelta delle classi è stata influenzata da 3 principi cardini che hanno consentito una scelta bilanciata tra la complessità dell'analisi e l'utilizzo ottimale dei tool presentati nel corso. Di seguito vengono presentate le 2 classi e le metriche scelte per la selezione.

BookKeeper

Come si evince nella *Tabella 1*, è possibile identificare un buon numero di valori di metriche importanti nell'ambito della *defect prediction* tra cui il *Number of Revisions*, le *LOC Touched* e, avendo dati disponibili al riguardo dal sistema JIRA anche la *buggynes* nella release presa in considerazione. Tali classi, come si può leggere anche dai successivi dati inclusi nello zip della presente relazione, sono spesso protagoniste di valori medio-alti anche nelle versioni successive, il motivo è probabilmente lapalissiano in quanto le classi scelte sono parte del "*kernel"* di BookKeeper:

- org.apache.bookkeeper.bookie.**EntryLogger**.java

  la scelta è stata condizionata da un alto numero [94] di revisioni che includevano modifiche alla classe in un'unica release, per una classe di 397 righe (all'epoca) sono state "toccate" ben 752 righe ovvero poco più del doppi delle righe che costituivano la stessa all'epoca dei *commit* inoltre, non potendocisi basare semplicemente su due metriche la scelta è stata condizionata anche dal ruolo centrale che la medesima ricopre nel progetto: essa è infatti responsabile della scrittura delle *entries* di bookkeeper e dell'apparato logistico per la ricerca e la restituzione delle entry attraverso un sistema di puntatori a posizioni (long) nei file gestiti dal logger e dalla **LedgerCache**.

- org.apache.bookkeeper.bookie.**Bookie**.java

  i parametri e le riflessioni effettuate per la precedente classe sono valide per la presente condividendo sia il numero alto di *commit* in una singola release che la riflessione sull'entità del valore delle LOC_Touched pari a circa il quintuplo delle righe della classe all'epoca del commit. La scelta è stata altresì motivata di nuovo dal ruolo della medesima nel progetto BookKeeper: i *bookie* rappresentano gli *storage server* i quali lavorano in modo distribuito al *matenimento* del contenuto dei Ledgers di BookKeeper.

Tajo

Come si può evincere dalla Tabella 2 anceh in questo caso le classi scelte posseggono interessanti valori sulle metriche collezionate durante la parte di progetto del Prof. Falessi, è utile sottolineare in effetti che anche in questo caso le due classi presentano valori di LOC_Touched pari quasi al doppio delle loro reali dimensioni inoltre data la natura di DataWarehouse di Tajo ho ritenuto le classi che gestiscono uno degli aspetti chiave di Tajo, ovvero le query, potessero essere degne di un'analisi più approfondita ed in effetti anche la mutation coverage effettuata con PitPM ha dato risultati interessati non solo sulla robustezza dei test scritti ma quanto più sulla robustezza dell'*engine* di query il quale è stato in grado di "resistere" bene all'"attacco" dei mutanti effettuando diverse *kill* in posizioni critiche evidenza anche di un buon prodotto software.

- org.apache.tajo.client.QueryClientImpl.java

  per la specifica classe è interessante notare il numero di revisioni in una singola release, ben 159 Jira non aveva informazioni sulla buggyness della presente classe ma le metriche raccolte indicano in effetti una classe che è stata modificata con una certa frequenza inoltre la presente classe implementa l'interfaccia "QueryClient" realizzazione del client per il sistema di query, essa quindi si occupa della sottomissione, scheduling & reporting dell'esito delle interrogazioni fatte su Tajo lato client componente critica per un sistema siffatto la quale potrebbe essere facilmente *entrypoint* per difetti e/o attacchi all'implementazione in termini di sicurezza. Le mutazioni in effetti, come si avrà l'opportunità di sottolineare più avanti, hanno confermato la mia ipotesi iniziale di classe suscettibile a difetti e quindi d'interesse per il testing.

- org.apache.tajo.master.QueryManager.java

  in effetti anche la presente classe ha subito molte revisioni circa 42 in un'unica revisione e se precedentemente si è voluta analizzare l'implementazione dell'interfaccia di interrogazione di Tajo era quindi giusto spostare l'attenzione lato management delle query stesse e formulare in prima istanza le medesime riflessioni poi confermate dall'uso di Badua e PIT.

Nelle classi scelte si è deciso di selezionare come metriche di adeguatezza dei test le metriche, in ordine di applicazione,

- branch & statement coverage fornite dagli output di Jacoco
- data flow coverage fornita dall'output di Ba-dua
- mutation coverage fornita da PIT

inoltre è giusto sottolineare una scelta necessaria su cui si è riflettuto molto in prima istanza ovvero il bilanciamento tra quantità di metodi e qualità dell'analisi, la formula da me scelta è stata cercare appunto di bilanciare la quantità di metodi da testare all'interno delle classi scelte soffermandomi sui metodi più interessanti, magari leggermente più complessi ma che poi, in effetti si sono rivelati essere interessanti da testare specie con lo strumento di PIT il quale ha confermato ipotesi iniziali e ha consentito di irrobustire alcuni test sottolineando problematiche alle quali i medesimi autori dei progetti dovrebbero in effetti prendere in considerazione.

La relazione continuerà presentando brevemente i singoli metodi delle classi scelte con le relative *category partition,* rimandando al codice per l'implementazione degli stessi, verranno quindi discussi i risultati ed i miglioramenti introdotti in ordine dall'analisi di Jacoco, Badua e PIT.

Category Partition

N.B: tutti i test scritti sono di tipo monodimensionale

BookKeeper

Bookie

public SettableFuture<Boolean> **fenceLedger**(**long** ledgerId, **byte**[] masterKey)

Il metodo under test si occupa di confinare un determinato ledger, è un'operazione idempotente, dal momento in cui è stato confinato nessun client, esclusa *l'addRecovery*, può più scrivere sul ledger in questione.

- per il parametro di tipo long si è pensato di eseguire test sui valori al bordo ovvero:
  - { -1 , 0 , 1 }
- Per il parametro di tipo array di byte si è invece ritenuto opportuno, come suggerito dalle slide del corso, di tenere conto dei seguenti valori
  - { null, array vuoto, array contenente "validi" byte}

I test, per la cui implementazione si rimanda al codice allegato, si sono svolti sulle seguenti coppie, si vorrebbe far notare, inoltre, che per diversi test sono state create "entity" di test, ovvero classi java, quindi oggetti, che svolgono il ruolo di contenitori delle variabili al fine di creare test parametrizzati:

	ledgerId	masterKey	expectedValue
FenceLedgerAux(	-**1L**,	**null**,	**false**)
FenceLedgerAux(	**0L**,	**new byte**[]{},	**true**)
FenceLedgerAux(	**1L**,	"masterKey".getBytes(),	**true**)

Fatta eccezione il primo test il quale solleva un eccezione e quindi mi aspetto di ottenere un valore "falso" confermato dall'eccezione, le altre due implementazioni consentono una corretta esecuzione del metodo senza eccezioni poiché il metodo *under test* non effettua controlli di esistenza sul ledger id, in realtà il ledger può addirittura essere creato sul momento e fenced, inoltre la master key, per accedere a *password protected ledger*, può di fatti essere vuota.

public static LedgerStorage **mountLedgerStorageOffline**
(ServerConfiguration conf, LedgerStorage ledgerStorage)

Il metodo consente di montare un'istanza di LedgerStorage offline per utilizzo da shell o altri utilizzi interattivi da parte in un utente, per esperienze pregresse l'accesso a file/volumi merita spesso una piccola attenzione, spesso può accadere di dare per scontato strutture delle directory o locazioni di file che in un test di unità

potrebbe ovviamente NON essere replicato e quindi consentire una più attenta analisi del comportamento isolato del metodo.

- ServerConfiguration, parametro ricorrente e "complesso", il suo stato infatti è determinato da diverse entità complesse e determina il comportamento generale del server/nodo di bookekeeper si può ritenere soddisfacente per ora un test di partizione:
    - {null, new ServerConfiguration()}

  Successive analisi porteranno infatti a creare un nuovo test predisponendo l'oggetto di configurazione in uno stato particolare.
- LedgerStorage: parametro complesso, anche il suo stato è definito da diversi sotto oggetti, in particolare dalla documentazione si evince quanto segue:
    - {null, new LedgerStorage()}

I test implementati vedono il combinarsi nei seguenti modi dei parametri coinvolti così come da category partition:

- Null, null, false
- ServerConfiguration, null, true
- ServerConfiguration, LedgerStorage, true

Secondo i parametri di ingresso, ad eccezione dell'ovvia coppia (null,nulll) consentono una corretta esecuzione del metodo, ovvero, non solleva eccezioni. In effetti da un'attenta analisi del sorgente a riga 648 della classe under test, si evince che se non venisse passato un ledger, ovvero venga passato il puntatore a null, esso si crea un nuovo ledger a partire dal primo parametro.

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

La funzione consente la lettura da un determinato *ledger* di una determinata *entry*, i parametri di ingresso rappresentano esattamente questi due concetti la category partition è quindi presto fatta:

- {-1,0,1,ledgerID/entryID realmente esistenti}

I quali, secondo quanto ci aspettiamo danno origine ai seguenti test:

```
new ReadEntryAux(0L, 1L, false)
new ReadEntryAux(1L, 0L, false)
new ReadEntryAux(realdLedgerID, realEntryID, true))
```

Ovviamente nella realizzazione delle prime due istanze del test parametrizzato ci possiamo attendere, come sia logico pensare, un errore da parte del metodo, mentre nel terzo caso, ovvero il caso "valido" oltre al supporto dell'istanziazione di un ambiente di test più "completo", esemplificata dalla presenza di una classe chiamata BookKeeperClusterTestCase scritta dagli autori del progetto BookKeeper si è reso necessario lo studio, agevolato dalla documentazione online, della composizione di una entry. L'entry, in generale per BookKeeper è essenzialmente un dump binario dei seguenti valori:

- [header dimensioni messaggio] (4 byte)
- LedgerID (long 8 byte)
- EntryID (long 8 byte)
- Messaggio (byte[dim])

Il sistema di scrittura si occupa di aggiungere il campo in rosso mentre il campo in azzurro rappresenta l'oggetto restituito o passato dal o al metodo, la verifica della bontà dell'esecuzione del metodo è stata garantita in ultima istanza dalla tipica prassi di *read-after-write*, si è quindi proceduto a scrivere su di un ledger e testato il comportamento del metodo osservando la restituzione dei eguali byte scritti in precedenza

EntryLogger

```
long addEntryForCompaction(long ledgerId, ByteBuf entry)
```

Il metodo in questione si occupa di ricevere una entry su cui effettuare una compattazione, la struttura dati di compattazione è leggermente differente poiché tiene conto di diversi parametri concernenti la gestione della medesima compattazione. La category partition in questo caso non è banalissima in quanto, se non si fosse studiato la struttura dati non si sarebbero potuti neanche selezione le giuste combinazioni di test, la versione "valida" ovvero con un long dichiarato ed una ByteBuf correttamente configurato con i parametri azzurri di cui sopra ha la necessità di far combaciare il LedgerID dichiarato con quello contenuto nella bytebuf come mezzo di maggiore sicurezza per evitare che un messaggio destinato al LedgerB potesse essere erroneamente intercettato o assegnato al LedgerA, la category partition da quindi origine a quanto segue:

```
new ImmutableTriple<>(true,1234L,"content".getBytes())
```

```
new ImmutableTriple<>(false,-1L,"".getBytes())
new ImmutableTriple<>(false,1L,null)
```

Nel primo caso viene impiegata principalmente la medesima strategia di read-after-write vista nel precedente metodo, andando a leggere direttamente sul file collegato all'oggeto FileChannell proprio dell'EntryLogger per verificare una coretta srittura, negli altri casi il comportamento è stato evidentemente conforme a quanto ci si aspettava con ledger inesistenti o con ByteBuff nulli.

```
public ByteBuf internalReadEntry
(long ledgerId, long entryId, long location, boolean validateEntry)
```

Si è scelto il presente metodo, al posto di una semplice readEntry poiché si è ritenuto opportuno e di valore l'analisi del comportamento di lettura a basso livello dell'EntryLogger, con il presente metodo è inoltre possibile validare o meno l'entry ciò consente un ottimo controllo a grana fine sul processo di lettura. Dalla presenza di diversi long e di un arametro boolean la category partition è presto fatta:

- Long: {-1,0,1,ledgerId/entryId/location}
- Boolean: {T,F}

```
new InternalReadEntryAux(-1L, -1L, -1L, true, false),
new InternalReadEntryAux(0L, 1L, 0L, false, false),
new InternalReadEntryAux(1L, 0L, 1L, false, false),
new InternalReadEntryAux(1234L, 1L, 0L, true, true),
```

Come nei precedent casi di test evidenziati finora il comportamento atteso è stato confermato, lì dove sono stati dati valori inventati, negativi, il metodo non completa la sua esecuzione correttamente e cade in eccezione mentre nel caso "valido" il metodo si comporta come ci si aspetterebbe e viene verificata la scrittura con una successiva lettura. Il valore booleano non compromette l'esecuzione corretta dell'applicazione, la test suit è insensitiva al cambiamento del flag, ovviamente ne risentirà la coverage del metodo.

Tajo

QueryClientImpl

```
public ResultSet executeQueryAndGetResult(String sql)
```

Il presente metodo si occupa di eseguire ed ottenere i risultati di una interrogazione in sql su Tajo lato client essa è un metodo bloccante nel caso di Query con tipologia FETCH ("contenenti clausole come WHERE") questa *behaviour* è correttamente implementata ma per una apparente incompatibilità dell'OpenJDK e dell'ambiente di test scritto dagli autori di Tajo ed implementate in QueryTestCaseBase il test poteva stallare in attesa di un timeout incorrettamente settato ma sepolto nell'implementazione dell'ambiente di test non imputabile allo scrivente, il metodo, come anche un altro in Tajo è stato completante testato ed espanso anceh con jacoco ma commentato per evitare problemi con TravisCI ma ritengo giusto sottolineare questa piccola incompatibilità.

La category partition su di una stringa è presto detta: {null, "", "istruzionevalida")
I test implementati sono stati così costruiti:

`public void invalidSQLCase()`	Stringa sql con istruzione invalida, comportamento atteso confermato, il metodo va in eccezione
`public void nullCase()`	Stringa sql nulla, anche in questo caso il comportamento atteso è stato confermato con l'eccezione alzata dal metodo under test
`public void validCase()`	Stringa sql di inserimento su table precedentemente creata in fase di istanziazione del test.

```
public SubmitQueryResponse executeQuery(final String sql)
```

Il presente metodo esegue immediatamente la query e restituisce in effetti un'oggetto diverso dal precedente metodo infatti da questo metodo è possibile avere immediatamente lo stato dell'interrogazione. La chiamata non è bloccante i test sono stati anche in questo caso, eseguiti in maniera parametrica ma per ovvie ragioni di persistenza del database e di race conditions tra tests ho pensato di implementare i successivi test, come il metodo precedente, a singoli metodi abbandonando la forma in realtà molto più comoda dei test parametrizzati. La category partition è la medesima del precedente metodo e da origine ai seguenti test:

```
new ImmutablePair<>(Errors.ResultCode.OK,"SELECT * FROM table3;"),
new ImmutablePair<>(Errors.ResultCode.INTERNAL_ERROR,""),
new ImmutablePair<>(Errors.ResultCode.UNDEFINED_COLUMN,"INSERT INTO table3 values (2, A);"),
new ImmutablePair<>(null,null));
```

Il metodo nel complesso si comporta in maniera coerente con quanto ci si aspettava, nel primo test infatti passando una query valida viene ritornato lo stato "OK", mentre restituisce "INTERNAL_ERROR" nel caso di query vuota, solleva eccezione con query nulla, ed "UNDEFINED_COLUMN" con tipi di dati non coerenti con quanto dichiarato nella creazione della tabella, essendo la sola stringa parametro di testing non si è voluto procedere con l'enumerazione di tutti i possibili Result Codes avendo comunque già esaurito i test monodimensionali della category partition.

public QueryStatus **getQueryStatus**(QueryId queryId)
Il metodo under test si occupa di restituire al chiamante lo stato della query il cui id è stato passato come parametro d'ingresso al metodo medesimo. Il tipo di dato QueryId è un tipo di dato complesso che prevede uno stato ben definito dai suoi attributi interni e la cui partizione è data come nei precedenti casi da 2 istanze: {null, queryID "valida"} è quindi da domandarsi, come fatto in precedenza cosa significhi "valida", ovviaemnte nel contesto corente uan query id valida è una query restituita dal sistema ad esmepio inseguito ad una execute o ad una submit escluso il valore q_00000 il quale è legato ad un resul_set vuoto, i test implementati sono i seguenti:

public invalidQueryId()	Il metodo si comporta come atteso, con una query non valida il metodo solleva un eccezione quindi nessun comportamento anomalo riscontrato
public validQueryId()	Nel caso di una queryId valida il metodo correttamente restituisce lo stato della query associata all'id, essendo stata precedentemente eseguita è possibile verifica diretta della correttezza dell'oggetto di ritorno
public nullQueryId()	Nel caso di una query nulla il metodo si presenta robusto e solleva un eccezione come ci si aspettava quindi nessun comportamento anomalo neanche in questo caso

QueryManager

public QueryInfo **getFinishedQuery**(QueryId queryId)
Il metodo under test in questo caso restituisce le informazioni di una query se e solo se essa è registrata come terminata, il valore potrebbe essere contenuto nella cache recente o nella cache globale del nodo master, per quanto concerne il partizionamento del tipo QueryId è sempre definito come {null, queryId} i test così indotti sono i seguenti:

public void validQueryID()	Nel caso di una QueryId valida il metodo correttamente restituisce lo stato della query associata all'id, nell'attuale contesto per "valida" si intende un QueryId associato ad una interrogazione terminata
public void erroneusSQLQueryID()	Il metodo si comporta come atteso, con una query non valida il metodo solleva un'eccezione quindi nessun comportamento anomalo riscontrato
public void nullQueryID()	Nel caso di una query nulla il metodo solleva un eccezione come ci si aspettava quindi nessun comportamento anomalo neanche in questo caso.

public QueryInProgress **getQueryInProgress**(QueryId queryId)
Il metodo under test in questo caso restituisce le informazioni di una query se e solo se essa è registrata come "*in progress*" o *"running"*, il metodo interroga la propria struttura dati alla ricerca dell'id qui specificato altrimeni, per quanto concerne il partizionamento del tipo QueryId è sempre definito come {null, queryId} i test così indotti sono i seguenti:

public void testInProgressQuery()	Il presente metodo testa il corretto funzionamento della chiamata al metodo under test con una query in progress, il metodo risponde correttamente
public void nullQuery()	Nel caso di una query nulla il metodo solleva un eccezione come ci si aspettava quindi nessun comportamento anomalo neanche in questo caso.
public void stoppedQuery()	Nel caso di una query stopped il metodo ritorna un puntatore a null, comportamento corretto e prevedibile

public void serviceInit(Configuration conf)
Il metodo under test si occupa di verificare il corretto funzionamento del metodo si inizializzazione del Master di Tajo è stato scelto per una questione di esperienza personale, spesso i metodi di inizializzazione possono nascondere in effetti delle soprese interessanti ed inoltre è interessante osservare come vengono gestite le fasi di inizializzazione di un progetto complesso come Tajo. Il comportamento del metodo è stato costante e prevedibile. Similmente a quanto detto per la Server Configuration di Bookie, anche per tajo, ad eccezione di stati particolari, la partizione è semplicemente data da: {null,valida} e nel contesto corrente per valida si intende una istanza di Configuration che consente l'inizializzazione del master di Tajo senza tenere conto di particolari stati delle entità interne. La partizione di categoria induce allora i seguenti test sul metodo

new ImmutablePair<>(**true**,conf)
new ImmutablePair<>(**false**,null)
In entrambe le occasioni il metodo è rimasto stabile ed il suo comportamento in linea con quanto deducibile dal codice e dalla logica, nel caso in cui si passi una configurazione "valida" il Master viene correttamente instanziato altrimenti ecceziona.

public void stopQuery(QueryId queryId)
il metodo, ultimo della presente analisi, si occupa di interrompere l'esecuzione di una query catalogata come "in progress" e di aggiornare i diversi oggetti coinvolti nella gestione delle query, come la history cache ec… solleva eccezione se al momento della richiesta di stop della query non sia stata rilevata nessuna query in progress. La category partition riferita all'oggetto queryId continua ad essere identica alle precedenti ed induce i seguenti test:

public void validQueryId()	Nel caso in qui venga passata una queryId "valida" il metodo termina correttamente, non solleva eccezioni ed è possibile controllare esternamente questa behaviour attraverso una successiva interrograzione alla lsita delle query in progress. Nel contesto corrente quindi il "valida" prende il signififcato di query running
public void nullQueryId()	Nel caso di una query nulla il metodo solleva un eccezione come ci si aspettava quindi nessun comportamento anomalo neanche in questo caso.
public void finishedQueryId()	Nel caso di una query stoppata il metodo termina senza side-effect e senza eccezioni

Qui si conclude la prima sezione del report, la category partition in effetti ha coperto una buona parte delle possibili combinazioni dei metodi, l'analisi successiva in realtà confermerà anche che in alcuni casi la category partition è stata sufficiente per raggiungere livelli di adeguatezza legati allo *statement coverage* o al *data flow coverage*

Adeguatezza e miglioramento dei test

Statement & Branch coverage JACOCO

Nel seguito verranno presentati i risultati e le modifiche apportate ai test case mantenendo un equilibrio tra precisione analitica e sintesi per evitare di effettuare *climanem* e perdere poi il fuoco della presente sezione il quale è incentrato nell'utilizzo dei tool presentati nel corso delle lezioni sull'adeguatezza dei test.

BookKeeper

Bookie

- **FenceLedger**: dalla natura semplice del metodo preso in considerazione jacoco ha evidenziato una coverage di statement del 100% si conclude quindi qui la trattazione del presente metodo come sottolineato dallo Snippet 1 per quanto riguarda la coverage
- **mountLedgerStorageOffline**: come evidenziato dallo Snippet 2 Jacoco data v1 per mountLedgerStorageOffline si può notare un branch non coperto a riga 648 poiché non gli veniva mai passato una ServerConfiguration valida con un ledgerStorage nullo, quindi è stato aggiunta una nuova entry all'array del test a riga 48 **new MountLedgerStorageOfflineAux**(**true**, **false**, **true**,**true**) il test ha quindi assicurato al metodo una coverage del 100% sia per il branch che per lo statement come evidenziato dallo Snippet 3 Jacoco data v2 per mountLedgerStorageOffline

- **readEntry:** l'analisi di jacoco, così come evidenziata dallo Snippet 4 Jacodo data per readEntry sopno evidenziati due branch non presi durante la run, o meglio, non sono stati scelti, durante la run, 2 combinazioni ovvero success == false & registerSuccesfull ed il contrario !succes & regsiterUnsuccesfull, dopo un'attenta analisi però si evince che non sia fattibile per la natura del metodo avere queste combinazioni *covered* in quanto la *try* impedisce l'arrivo al valore success=true e quindi impedisce quel path e vice versa, in caso di assenza di eccezioni no è possibile, dall'esterno configurare gli oggetti per indure queste path perciò non sono stati aggiunti test al riguardo e per tale motivo.

EntryLogger

- **addEntryForCompaction**: come si può notare dallo Snippet 5 Jacoco data v1 per addEntryForCompation esiste anche in questo metodo un branch non coperto a riga 619, in effetti è possibile coprire questo branch con il test aggiunto alla riga 54 `new ImmutableTriple<>(true,12345L,"content".getBytes())` in effetti con l'aggiunta di questo elemento all'array del test come sottolinea lo Snippet 6 Jacoco data v2 for addEntryForCompaction si ottiene lo statement coverage ed il branch coverage pari al 100%
- **internalReadEntry:** è possibile osservare nello Snippet 7 Jacoco data v1 for internalReadEntry che anche in questo caso Jacoco ha scoperto una combinazione di salti non presi è stato sufficiente aggiungere un test per coprire uno dei due salti, il secondo non poteva essere coperto con facilità senza una tecnica più time-consuming con le mock, il test aggiunto ha consentito una coverage migliorata ed un test più adattato a testare il metodo così come evidenziato dallo Snippet 8 Jacoco data v2 for internalReadEntry

Tajo

QueryClienImpl

- **executeQueryAndGetResult:** come evidenziato dallo Snippet 9 Jacoco data v1 per executeQueryAndGetResult 2 casi dello switch a riga 199 non sono stati coperti dai precedenti test sono stati quindi aggiunti due nuovi test alle righe 67 e 79 portando il metodo al 100% di coverage, ma il test, come già discusso nella sezione precedente di category partition il test non era *reliable* a causa di incompatibilità tra l'ambiente di test ed il jdk perciò se pur coperte in realtà al 100 % riporto il dato con il metodo commentato così come sottolineato dalla repository e dallo Snippet 10 Jacoco data v2 per executeQueryAndGetResult
- **executeQuery:** è possibile evincere dalla lettura dei dati dello Snippet 11 Jacoco data per executeQuery che il metodo gode di una branch coverage del 100% mentre per lo *statement* vale quanto detto per il metodo internalReadEntry
- **getQueryStatus**: dall'analisi dello Snippet 12 Jacoco data per getQueryStatus si evince in effetti la medesima realizzazione del precedente metodo

QueryManagemenet

- **getFinishedQuery**: lo Snippet 13 Jacoco data per getFinishedQuery evidenza una situazione molto simile ai precedenti metodi ovvero una branch del 100% mentre una *statement* del 75% le eccezioni di tipo RuntimeException sono copribili solo con attenta e analitica mock di varie entità al fine di creare un trigger per l'eccezione, come negli altri casi il tempo è stato tiranno altrimenti con paicere sarebbe stato molto interessante creare questo triggger.
- **getQueryInProgress:** si può evincere dallo Snippet 14 Jacoco datra per getQueryInProgress un metodo con coverage al 100% sia da un punto di vista di statement che di branch non è quindi necessario commentare ulteriormente questo metodo
- **serviceInit:** come si può notare nello Snippet 16 Jacoco data per serviceInit il metodo, data anche la sua estrema semplicità ha il 100% di coverage branch & statement.
- **stopQuery:** per il metodo in questione il discorso è leggermente diverso, possiede ottimi valori di coverage come si può evincere dallo Snippet 15 Jacoco data per stopQuery rispettivamente 85% di *statement* e 62% di branch ma gli oggetti al contorno necessitano di un'analisi più approfondita e di un controllo maggiore della classe attraverso mock, infatti minExecutionTime è un attributo privato della classe privo di getter & setter quindi inalterabile dall'esterno se non dall'interno assumendo maggiore controllo dell'oggetto

BookKeeper

Bookie

- **FenceLedger**: <no-data>
- **mountLedgerStorageOffline**: dall'analisi del report di badua e dello Snippet 17 Ba-Dua data v1 coverage per mountLedgerStorageOffline si evince che il metodo, in prima analisi, non possedeva una data flow coverage completa ma se ricordiamo anche il valore di riferimento per il code coverage non era del 100% in seguito all'aggiunta dei test descritti nella sezione di Jacoco si osserva una rapida mutazione dei valori anche in Ba-Dua la quale registra alla fine una data coverage del 100% così come riportato nello Snippet 18 Ba-Dua data v2 coverage per mountLedgerStorageOffline
- **readEntry:** pur essendo un tool in via di sviluppo e quindi sensibile a condizioni che altri tool più maturi come lo stesso Jacoco ignorano, dall'analisi del primo report di cui allo Snippet 19 Ba-Dua data v0 readEntry questo metodo risultava avere 0 accessi come se fosse stato completamente saltato, invece da un'attenta analisi con debugger è stato possibile evidenziare un errore implementativo del test il quale, nonostante funzionasse sollevasse un'eccezione sul finire del primo test che JUnit ignora mentre a Ba-dua andava a sua volta in eccezione in quel punto e saltava il metodo, al termine di questa scoperta è stato possibile osservare i parametri di cui allo Snippet 20 Ba-dua data per readEntry, rimandano al codice si evidenzia che alcuni archi evidenziati dal Badua come non coperti siano in realtà coperti dall'evidenza empirica dei test, come ad esempio l'arco 1424->1442 che coinvolge la variabile NANOSECONDS, esso in realtà è conforme e coperto al medesimo modo dell'arco segnato invece come coperto 1424->1439. Nel caso specifico come più volte specificato in precedenza non avendo potere su variabili private inaccessibili non mi è stato possibile migliorare la data coverage del metodo a differenza del precedente.

EntryLogger

- **addEntryForCompaction**: anche in questo caso una esecuzione simile a readEntry ha fatto notare un'eccezione non correttamente gestita e a seguito di alcuni aggiustamenti come evidenziato anche dallo stesso Jacoco sulla coverage anche lo Snippet 21 Ba-Dua data per addEntryForCompaction conferma la bontà dei test riportando un risultato ottimale, ovvero copertura completa
- **internalReadEntry:** il test case, secondo I dati di Ba-dua si presentava effettivamente ben coperto dall'attuale punto di vista del Data Flow Coverage come riportato dallo Snippet 23 Ba-Dua data v1 per internalReadEntry in seguito ai miglioramenti effettuati sulla base delle informazioni di jacoco e di Ba-dua è stato possibile migliorare ancora di più la coverage così come da Snippet 22 Ba-Dua data v2 per internalReadEntry

Tajo

QueryClienImpl

- **executeQueryAndGetResult:** come è possibile osservare dal report di Ba-Dua Snippet 24 Ba-dua data per executeQueryAndGetResult il test case conferma la sua bontà nella realtà di codice implementato per il test si rimanda alla sezione precedente ove specificavo che per via di incompatibilità di jdk è stato necessario commentare un test e quindi oltre alla code coverage che impedisce di arrivare al 100% ovviamente anche la data flow covergae ne risnete non potendo arrivare al 100%
- **executeQuery:** si può evincere che il test case, complice anche la semplicità del metodo testato, è coperto al 100% anche dal punto di vista di Data Flow Coverage così come sottolineato dallo Snippet 25 Ba-dua data per executeQuery
- **getQueryStatus**: <no-data>

QueryManagemenet

- **getFinishedQuery**: come è possibile ossevrare dal report di badua djks la bontà del test case è confermata da una coverage del 100% anche dal punto di vista del Data Flow Coverage
- **getQueryInProgress:** il presente metodo non era coperto completamente dal punto di vista del Data Flow Coverage come è evidenziato dallo Snippet 27 Ba-dua per getQueryInProgress migliorato anche grazie ai dati di Jacoco il report finale di Ba-dua infatti sottolinea una coverage del 100% Snippet 28 Ba-dua data v2 per getQueryInProgress

- **serviceInit:** come evidenziato anche nella sezione di jacoco non è posisbile avere coverage al 100% per via dello stato degli oggetti interni privati ne risente la Data Flow Coverage infatti non è possibile ottenere più del 90% circa come riportato dallo Snippet 29 Ba-dua data per serviceInit
- **stopQuery:** similmente ad altri metodi, inzialmente Ba-dua asseriva che il metodo fosse totalmente saltato mentre al temrine dei fixes il metodo risulta coperto al 90% solo 2 archi non sono stati coperti così come è evidente dallo Snippet 30 Ba-dua data per stopQuery.

Adeguatezza e miglioramento dei test

Mutation coverage PIT & PitMP

BookKeeper

Bookie

47% 289/616 22% 50/232

- **FenceLedger**: il test case si è rivelato essere robusto contro le mutazioni di pit killando la mutazione generata come evidenziato dallo Snippet 31 Pit data per fenceLedger
- **mountLedgerStorageOffline**: in questo caso il test case, anche in relazione a quanto detto nelle sezioni precedenti sul poco controllo di alcuni aspetti della classe, si è rivelato egualmente robusto ma 3 mutazioni sono sopravvissute come si può evincere Snippet 32 Pit data per per mountLedgerStorageOffline in seguito alla lettura dei presenti dati è stato possibile migliorare la mutation coverage come evidenziato Snippet 36 Pit data v2 per mountLedgerStorageOffline
- **readEntry:** il test case risulta essere mediamente robusto alle mutazioni introdotte, per via della complessità della cattura di determinate mutazioni, solito discorso sul controllo della parte privata dello stato della classe i dati riportati sono rimasti invariati rispetto all'analisi finale post ba-dua e jacoco riportati nello Snippet 33 Pit data per readEntry

EntryLogger

41% 150/368 16% 24/154

- **addEntryForCompaction**: come per il caso di readEntry e come seguirà per internalRead Entry i repo non sono migliorati rispetto all'ultima analisi post badua & Jacoco fixes come evidenziato dallo Snippet 34 Pit data per addEntryForCompaction
- **internalReadEntry:** il report è contenuto nello Snippet 35 Pit data per internalReadEntry

Tajo

In relazione al progetto Tajo, la posizione obbligata dei test sotto il modulo "Tajo-cluster-tests" ha reso il run di PIT molto più complesso attraverso la sua estensione multi progetto PitMP! Inoltre la memoria che il nuovo plug-in allocava nella jvm risultava leggermente difettata e spesso crashava quindi sono stato costretto a restringere le operazioni di mutazione ai valori di ritorno, i dati della prima run sono andati persi in un mvn clean sono quindi qui presentati solamente i dati finali comprensivi dei miglioramenti che verranno descritti senza lo screenshot della precedente versione comunque individuabili attraverso i commit sulla repository.

QueryClienImpl

27% 65/245 14% 6/44

- **executeQueryAndGetResult:** rispetto alla generazione precendente sono riuscito a killare una mutazione così come sotolineato dai commit, il report è presente nello Snippet 38 Pit data per exeeceuteQueryAndGetResult
- **executeQuery:** il test case è risultato robusto rispetto alla mutazione come dimostrato dallo Snippet 37 Pit data per executeQuery
- **getQueryStatus**: come per executeQuery, il test case è risultato robusto rispetto alla mutazione come dimostrato dallo Snippet 39 Pit data per getQueryStatus

QueryManagemenet

56% 80/142 21% 5/24

- **getFinishedQuery**: il metodo presenta due mutazioni una killata mentre la seconda sopravvissuta
- **getQueryInProgress:** il test case è risultato essere robusto rispetto alla mutazione introdotta come evidenziato dallo Snippet 41Pit data per QueryInProgress
- **serviceInit:** <no-mutations-generated> - Snippet 42 Pit data per serviceInit
- **stopQuery:** <no-mutations-generated> - Snippet 43 Pit data per StopQuery

Considerazioni finali

Il progetto è stato molto interessante, operare su un insieme di progetti non propri e complessi ovviamente presenta delle difficoltà intrinseche ma ritengo di aver usufruito di questa possibilità per raffinare le conoscenze delle piattaforme di Travis-CI e SonarCloud e l'importanza che ricade non solo nel puro testing bensì nella ricerca puntuale dell'adeguatezza dei test scritti. Tale punto di vista è essenziale per consegnare progetti software di buon livello e robusti. La posizione dei test poco ortodossa, nel caso di bookkeeper in BookKeeper-Server e nel caso di Tajo sotto Tajo-cluster-tests, sono state dettate da possibilità di accedere a oggetti package protected o per evitare dipendenze circolari.

Materiale Incluso

Tabelle

LOC	AVG_Chur	NR	AVG_LOC_added	ChgSetSize	AVG_ChgS	Churn	LOC_added	MAX_Churn	Version	MAX_LOC_added	MAX_ChgSet	File Name	LOC_touched	NAuth	WeightedAge	Age	Buggy
397	71	94	80	65787868	14430	1005	1130	545	release-4.0.0	545	21081	Bookie.java	1350	3	330750	245	YES
364	61	94	65	65787868	14430	551	590	487	release-4.0.0	487	21081	EntryLogger.java	752	3	184240	245	YES

Tabella 1

(le metriche basate su LOC escludono i commenti)

LOC	AVG_Churn	NR	AVG_LOC_added	ChgSetSize	AVG_ChgSet	Churn	LOC_addec	MAX_Churn	Version	MAX_LOC_added	MAX_ChgSet	File Name	LOC_touched	NAuth	VeightedAg	Age	Buggy
404	75	159	70	1869378674	121207	755	700	622	release-0.10.0	622	231642	QueryClientImpl.java	807	8	95226	118	ND
200	60	42	58	865647025	132544	360	351	315	release-0.10.0	315	231642	QueryManager.java	367	6	16515	45	ND

Tabella 2

Snippet of Code

```
409.     /**
410.      * Fences a ledger. From this point on, clients will be unable to
411.      * write to this ledger. Only recoveryAddEntry will be
412.      * able to add entries to the ledger.
413.      * This method is idempotent. Once a ledger is fenced, it can
414.      * never be unfenced. Fencing a fenced ledger has no effect.
415.      */
416.     public SettableFuture<Boolean> fenceLedger(long ledgerId, byte[] masterKey)
417.             throws IOException, BookieException {
418.         LedgerDescriptor handle = handles.getHandle(ledgerId, masterKey);
419.         return handle.fenceAndLogInJournal(getJournal(ledgerId));
420.     }
421.
```

Snippet 1 Jacoco data per metodo FenceLedger

```
627.     /**
628.      * Initialize LedgerStorage instance without checkpointing for use within the shell
629.      * and other RO users.  ledgerStorage must not have already been initialized.
630.      *
631.      * <p>The caller is responsible for disposing of the ledgerStorage object.
632.      *
633.      * @param conf Bookie config.
634.      * @param ledgerStorage Instance to initialize.
635.      * @return Passed ledgerStorage instance
636.      * @throws IOException
637.      */
638.     public static LedgerStorage mountLedgerStorageOffline(ServerConfiguration conf, LedgerStorage ledgerStorage)
639.             throws IOException {
640.         StatsLogger statsLogger = NullStatsLogger.INSTANCE;
641.         DiskChecker diskChecker = new DiskChecker(conf.getDiskUsageThreshold(), conf.getDiskUsageWarnThreshold());
642.
643.         LedgerDirsManager ledgerDirsManager = createLedgerDirsManager(
644.                 conf, diskChecker, statsLogger.scope(LD_LEDGER_SCOPE));
645.         LedgerDirsManager indexDirsManager = createIndexDirsManager(
646.                 conf, diskChecker, statsLogger.scope(LD_INDEX_SCOPE), ledgerDirsManager);
647.
648.         if (null == ledgerStorage) {
649.             ledgerStorage = buildLedgerStorage(conf);
650.         }
651.
652.         CheckpointSource checkpointSource = new CheckpointSource() {
653.             @Override
654.             public Checkpoint newCheckpoint() {
655.                 return Checkpoint.MAX;
656.             }
657.
658.             @Override
659.             public void checkpointComplete(Checkpoint checkpoint, boolean compact)
660.                     throws IOException {
661.             }
662.         };
```

Snippet 2 Jacoco data v1 per mountLedgerStorageOffline

```
627.    /**
628.     * Initialize LedgerStorage instance without checkpointing for use within the shell
629.     * and other RO users.  ledgerStorage must not have already been initialized.
630.     *
631.     * <p>The caller is responsible for disposing of the ledgerStorage object.
632.     *
633.     * @param conf Bookie config.
634.     * @param ledgerStorage Instance to initialize.
635.     * @return Passed ledgerStorage instance
636.     * @throws IOException
637.     */
638.    public static LedgerStorage mountLedgerStorageOffline(ServerConfiguration conf, LedgerStorage ledgerStorage)
639.            throws IOException {
640.        StatsLogger statsLogger = NullStatsLogger.INSTANCE;
641.        DiskChecker diskChecker = new DiskChecker(conf.getDiskUsageThreshold(), conf.getDiskUsageWarnThreshold());
642.
643.        LedgerDirsManager ledgerDirsManager = createLedgerDirsManager(
644.                conf, diskChecker, statsLogger.scope(LD_LEDGER_SCOPE));
645.        LedgerDirsManager indexDirsManager = createIndexDirsManager(
646.                conf, diskChecker, statsLogger.scope(LD_INDEX_SCOPE), ledgerDirsManager);
647.
648.        if (null == ledgerStorage) {
649.            ledgerStorage = buildLedgerStorage(conf);
650.        }
651.
652.        CheckpointSource checkpointSource = new CheckpointSource() {
653.            @Override
654.            public Checkpoint newCheckpoint() {
655.                return Checkpoint.MAX;
656.            }
657.
658.            @Override
659.            public void checkpointComplete(Checkpoint checkpoint, boolean compact)
660.                    throws IOException {
661.            }
662.        };
```

Snippet 3 Jacoco data v2 per mountLedgerStorageOffline

```
1422.    public ByteBuf readEntry(long ledgerId, long entryId)
1423.            throws IOException, NoLedgerException {
1424.        long requestNanos = MathUtils.nowInNano();
1425.        boolean success = false;
1426.        int entrySize = 0;
1427.        try {
1428.            LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1429.            if (LOG.isTraceEnabled()) {
1430.                LOG.trace("Reading {}@{}", entryId, ledgerId);
1431.            }
1432.            ByteBuf entry = handle.readEntry(entryId);
1433.            bookieStats.getReadBytes().add(entry.readableBytes());
1434.            success = true;
1435.            return entry;
1436.        } finally {
1437.            long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1438.            if (success) {
1439.                bookieStats.getReadEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1440.                bookieStats.getReadBytesStats().registerSuccessfulValue(entrySize);
1441.            } else {
1442.                bookieStats.getReadEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1443.                bookieStats.getReadEntryStats().registerFailedValue(entrySize);
1444.            }
1445.        }
1446.    }
1447.
```

Snippet 4 Jacoco data per readEntry

```
616.     long addEntryForCompaction(long ledgerId, ByteBuf entry) throws IOException {
617.         synchronized (compactionLogLock) {
618.             int entrySize = entry.readableBytes() + 4;
619.             if (compactionLogChannel == null) {
620.                 createNewCompactionLog();
621.             }
622. 
623.             ByteBuf sizeBuffer = this.sizeBuffer.get();
624.             sizeBuffer.clear();
625.             sizeBuffer.writeInt(entry.readableBytes());
626.             compactionLogChannel.write(sizeBuffer);
627. 
628.             long pos = compactionLogChannel.position();
629.             compactionLogChannel.write(entry);
630.             compactionLogChannel.registerWrittenEntry(ledgerId, entrySize);
631.             return (compactionLogChannel.getLogId() << 32L) | pos;
632.         }
633.     }
```

Snippet 5 Jacoco data v1 per addEntryForCompation

```
616.     long addEntryForCompaction(long ledgerId, ByteBuf entry) throws IOException {
617.         synchronized (compactionLogLock) {
618.             int entrySize = entry.readableBytes() + 4;
619.             if (compactionLogChannel == null) {
620.                 createNewCompactionLog();
621.             }
622. 
623.             ByteBuf sizeBuffer = this.sizeBuffer.get();
624.             sizeBuffer.clear();
625.             sizeBuffer.writeInt(entry.readableBytes());
626.             compactionLogChannel.write(sizeBuffer);
627. 
628.             long pos = compactionLogChannel.position();
629.             compactionLogChannel.write(entry);
630.             compactionLogChannel.registerWrittenEntry(ledgerId, entrySize);
631.             return (compactionLogChannel.getLogId() << 32L) | pos;
632.         }
633.     }
```

Snippet 6 Jacoco data v2 for addEntryForCompaction

```
816.    public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry)
817.            throws IOException, Bookie.NoEntryException {
818.        long entryLogId = logIdForOffset(location);
819.        long pos = posForOffset(location);
820.
821.
822.        BufferedReadChannel fc = null;
823.        int entrySize = -1;
824.        try {
825.            fc = getFCForEntryInternal(ledgerId, entryId, entryLogId, pos);
826.
827.            ByteBuf sizeBuff = readEntrySize(ledgerId, entryId, entryLogId, pos, fc);
828.            entrySize = sizeBuff.getInt(0);
829.            if (validateEntry) {
830.                validateEntry(ledgerId, entryId, entryLogId, pos, sizeBuff);
831.            }
832.        } catch (EntryLookupException.MissingEntryException entryLookupError) {
833.            throw new Bookie.NoEntryException("Short read from entrylog " + entryLogId,
834.                    ledgerId, entryId);
835.        } catch (EntryLookupException e) {
836.            throw new IOException(e.toString());
837.        }
838.
839.        ByteBuf data = allocator.buffer(entrySize, entrySize);
840.        int rc = readFromLogChannel(entryLogId, fc, data, pos);
841.        if (rc != entrySize) {
842.            // Note that throwing NoEntryException here instead of IOException is not
843.            // without risk. If all bookies in a quorum throw this same exception
844.            // the client will assume that it has reached the end of the ledger.
845.            // However, this may not be the case, as a very specific error condition
846.            // could have occurred, where the length of the entry was corrupted on all
847.            // replicas. However, the chance of this happening is very very low, so
848.            // returning NoEntryException is mostly safe.
849.            data.release();
850.            throw new Bookie.NoEntryException("Short read for " + ledgerId + "@"
851.                                              + entryId + " in " + entryLogId + "@"
852.                                              + pos + "(" + rc + "!=" + entrySize + ")", ledgerId, entryId);
853.        }
854.        data.writerIndex(entrySize);
855.
856.        return data;
857.    }
```

Snippet 7 Jacoco data v1 for internalReadEntry

```
816.    public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry)
817.            throws IOException, Bookie.NoEntryException {
818.        long entryLogId = logIdForOffset(location);
819.        long pos = posForOffset(location);
820.
821.
822.        BufferedReadChannel fc = null;
823.        int entrySize = -1;
824.        try {
825.            fc = getFCForEntryInternal(ledgerId, entryId, entryLogId, pos);
826.
827.            ByteBuf sizeBuff = readEntrySize(ledgerId, entryId, entryLogId, pos, fc);
828.            entrySize = sizeBuff.getInt(0);
829.            if (validateEntry) {
830.                validateEntry(ledgerId, entryId, entryLogId, pos, sizeBuff);
831.            }
832.        } catch (EntryLookupException.MissingEntryException entryLookupError) {
833.            throw new Bookie.NoEntryException("Short read from entrylog " + entryLogId,
834.                    ledgerId, entryId);
835.        } catch (EntryLookupException e) {
836.            throw new IOException(e.toString());
837.        }
838.
839.        ByteBuf data = allocator.buffer(entrySize, entrySize);
840.        int rc = readFromLogChannel(entryLogId, fc, data, pos);
841.        if (rc != entrySize) {
842.            // Note that throwing NoEntryException here instead of IOException is not
843.            // without risk. If all bookies in a quorum throw this same exception
844.            // the client will assume that it has reached the end of the ledger.
845.            // However, this may not be the case, as a very specific error condition
846.            // could have occurred, where the length of the entry was corrupted on all
847.            // replicas. However, the chance of this happening is very very low, so
848.            // returning NoEntryException is mostly safe.
849.            data.release();
850.            throw new Bookie.NoEntryException("Short read for " + ledgerId + "@"
851.                                              + entryId + " in " + entryLogId + "@"
852.                                              + pos + "(" + rc + "!=" + entrySize + ")", ledgerId, entryId);
853.        }
854.        data.writerIndex(entrySize);
855.
856.        return data;
857.    }
```

Snippet 8 Jacoco data v2 for internalReadEntry

```
191.  @Override
192.  public ResultSet executeQueryAndGetResult(String sql) throws TajoException {
193.
194.    SubmitQueryResponse response = executeQuery(sql);
195.    throwIfError(response.getState());
196.
197.    QueryId queryId = new QueryId(response.getQueryId());
198.
199.    switch (response.getResultType()) {
200.      case ENCLOSED:
201.        return TajoClientUtil.createResultSet(this, response, defaultFetchRows);
202.      case FETCH:
203.        return this.getQueryResultAndWait(queryId);
204.      default:
205.        return this.createNullResultSet(queryId);
206.    }
207.  }
```

Snippet 9 Jacoco data v1 per executeQueryAndGetResult

```
191.  @Override
192.  public ResultSet executeQueryAndGetResult(String sql) throws TajoException {
193.
194.    SubmitQueryResponse response = executeQuery(sql);
195.    throwIfError(response.getState());
196.
197.    QueryId queryId = new QueryId(response.getQueryId());
198.
199.    switch (response.getResultType()) {
200.      case ENCLOSED:
201.        return TajoClientUtil.createResultSet(this, response, defaultFetchRows);
202.      case FETCH:
203.        return this.getQueryResultAndWait(queryId);
204.      default:
205.        return this.createNullResultSet(queryId);
206.    }
207.  }
```

Snippet 10 Jacoco data v2 per executeQueryAndGetResult

```
158.  @Override
159.  public SubmitQueryResponse executeQuery(final String sql) {
160.
161.    final BlockingInterface stub = conn.getTMStub();
162.    final QueryRequest request = buildQueryRequest(sql, false);
163.
164.    SubmitQueryResponse response;
165.    try {
166.      response = stub.submitQuery(null, request);
167.    } catch (ServiceException e) {
168.      throw new RuntimeException(e);
169.    }
170.
171.    if (isSuccess(response.getState())) {
172.      conn.updateSessionVarsCache(ProtoUtil.convertToMap(response.getSessionVars()));
173.    }
174.
175.    return response;
176.
177.  }
```

Snippet 11 Jacoco data per executeQuery

```
270.   @Override
271.   public QueryStatus getQueryStatus(QueryId queryId) throws QueryNotFoundException {
272.
273.     final BlockingInterface stub = conn.getTMStub();
274.     final GetQueryStatusRequest request = GetQueryStatusRequest.newBuilder()
275.         .setSessionId(conn.sessionId)
276.         .setQueryId(queryId.getProto())
277.         .build();
278.
279.     GetQueryStatusResponse res;
280.     try {
281.       res = stub.getQueryStatus(null, request);
282.     } catch (ServiceException t) {
283.       throw new RuntimeException(t);
284.     }
285.
286.     throwsIfThisError(res.getState(), QueryNotFoundException.class);
287.     ensureOk(res.getState());
288.     return new QueryStatus(res);
289.   }
```

Snippet 12 Jacoco data per getQueryStatus

```
172.   public QueryInfo getFinishedQuery(QueryId queryId) {
173.     try {
174.       QueryInfo queryInfo;
175.       synchronized (historyCache) {
176.         queryInfo = (QueryInfo) historyCache.get(queryId);
177.       }
178.       if (queryInfo == null) {
179.         queryInfo = this.masterContext.getHistoryReader().getQueryByQueryId(queryId);
180.       }
181.       return queryInfo;
182.     } catch (Throwable e) {
183.       LOG.error(e.getMessage(), e);
184.       return null;
185.     }
186.   }
```

Snippet 13 Jacoco data per getFinishedQuery

```
267.   public QueryInProgress getQueryInProgress(QueryId queryId) {
268.     QueryInProgress queryInProgress;
269.     queryInProgress = submittedQueries.get(queryId);
270.
271.     if (queryInProgress == null) {
272.       queryInProgress = runningQueries.get(queryId);
273.     }
274.
275.     return queryInProgress;
276.   }
```

Snippet 14 Jacoco datra per getQueryInProgress

```
278.  public void stopQuery(QueryId queryId) {
279.    LOG.info("Stop QueryInProgress:" + queryId);
280.    QueryInProgress queryInProgress = getQueryInProgress(queryId);
281.    if(queryInProgress != null) {
282.      queryInProgress.stopProgress();
283.      QueryInfo queryInfo = queryInProgress.getQueryInfo();
284.      synchronized (historyCache) {
285.        historyCache.put(queryInfo.getQueryId(), queryInfo);
286.      }
287.
288.      submittedQueries.remove(queryId);
289.      runningQueries.remove(queryId);
290.
291.      long executionTime = queryInfo.getFinishTime() - queryInfo.getStartTime();
292.      if (executionTime < minExecutionTime.get()) {
293.        minExecutionTime.set(executionTime);
294.      }
295.
296.      if (executionTime > maxExecutionTime.get()) {
297.        maxExecutionTime.set(executionTime);
298.      }
299.
300.      long totalExecutionTime = executedQuerySize.get() * avgExecutionTime.get();
301.      if (totalExecutionTime > 0) {
302.        avgExecutionTime.set((totalExecutionTime + executionTime) / (executedQuerySize.get() + 1));
303.      } else {
304.        avgExecutionTime.set(executionTime);
305.      }
306.      executedQuerySize.incrementAndGet();
307.    } else {
308.      LOG.warn("No QueryInProgress while query stopping: " + queryId);
309.    }
310.  }
```

Snippet 15 Jacoco data per stopQuery

```
79.  @Override
80.  public void serviceInit(Configuration conf) throws Exception {
81.    try {
82.      this.dispatcher = new AsyncDispatcher();
83.      addService(this.dispatcher);
84.
85.      this.dispatcher.register(QueryJobEvent.Type.class, new QueryJobManagerEventHandler());
86.
87.      TajoConf tajoConf = TUtil.checkTypeAndGet(conf, TajoConf.class);
88.      this.historyCache = new LRUMap(tajoConf.getIntVar(TajoConf.ConfVars.HISTORY_QUERY_CACHE_SIZE));
89.    } catch (Exception e) {
90.      LOG.error("Failed to init service " + getName() + " by exception " + e, e);
91.    }
92.
93.    super.serviceInit(conf);
94.  }
```

Snippet 16 Jacoco data per serviceInit

```
▼<method name="mountLedgerStorageOffline" desc="(Lorg/apache/bookkeeper/conf/ServerConfiguration;Lorg/apache/bookkeeper/bookie/LedgerStorage;)Lorg/apache/bookkeeper/bookie/LedgerStorage;">
    <du var="conf" def="640" use="666" covered="1"/>
    <du var="conf" def="640" use="649" covered="0"/>
    <du var="ledgerStorage" def="640" use="648" target="649" covered="0"/>
    <du var="ledgerStorage" def="640" use="648" target="652" covered="1"/>
    <du var="ledgerStorage" def="640" use="666" covered="1"/>
    <du var="ledgerStorage" def="640" use="677" covered="1"/>
    <du var="NULL" def="640" use="664" covered="1"/>
    <du var="DEFAULT" def="640" use="666" covered="1"/>
    <du var="statsLogger" def="640" use="666" covered="1"/>
    <du var="ledgerDirsManager" def="643" use="666" covered="1"/>
    <du var="indexDirsManager" def="645" use="666" covered="1"/>
    <du var="ledgerStorage" def="649" use="666" covered="0"/>
    <du var="ledgerStorage" def="649" use="677" covered="0"/>
    <counter type="DU" missed="4" covered="9"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>
```

Snippet 17 Ba-Dua data v1 coverage per mountLedgerStorageOffline

```
▼<method name="mountLedgerStorageOffline" desc="(Lorg/apache/bookkeeper/conf/ServerConfiguration;Lorg/apache/bookkeeper/bookie/LedgerStorage;)Lorg/apache/bookkeeper/bookie/LedgerStorage;">
  <du var="conf" def="640" use="666" covered="1"/>
  <du var="conf" def="640" use="649" covered="1"/>
  <du var="ledgerStorage" def="640" use="648" target="649" covered="1"/>
  <du var="ledgerStorage" def="640" use="648" target="652" covered="1"/>
  <du var="ledgerStorage" def="640" use="666" covered="1"/>
  <du var="ledgerStorage" def="640" use="677" covered="1"/>
  <du var="NULL" def="640" use="664" covered="1"/>
  <du var="DEFAULT" def="640" use="666" covered="1"/>
  <du var="statsLogger" def="640" use="666" covered="1"/>
  <du var="ledgerDirsManager" def="643" use="666" covered="1"/>
  <du var="indexDirsManager" def="645" use="666" covered="1"/>
  <du var="ledgerStorage" def="649" use="666" covered="1"/>
  <du var="ledgerStorage" def="649" use="677" covered="1"/>
  <counter type="DU" missed="0" covered="13"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```


Snippet 18 Ba-Dua data v2 coverage per mountLedgerStorageOffline


```
▼<method name="readEntry" desc="(JJ)Lio/netty/buffer/ByteBuf;">
  <du var="this" def="1424" use="1433" covered="0"/>
  <du var="this" def="1424" use="1442" covered="0"/>
  <du var="this" def="1424" use="1443" covered="0"/>
  <du var="this" def="1424" use="1439" covered="0"/>
  <du var="this" def="1424" use="1440" covered="0"/>
  <du var="ledgerId" def="1424" use="1430" covered="0"/>
  <du var="entryId" def="1424" use="1432" covered="0"/>
  <du var="entryId" def="1424" use="1430" covered="0"/>
  <du var="LOG" def="1424" use="1429" target="1430" covered="0"/>
  <du var="LOG" def="1424" use="1429" target="1432" covered="0"/>
  <du var="LOG" def="1424" use="1430" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1433" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1442" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1443" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1439" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1440" covered="0"/>
  <du var="NANOSECONDS" def="1424" use="1442" covered="0"/>
  <du var="NANOSECONDS" def="1424" use="1439" covered="0"/>
  <du var="requestNanos" def="1424" use="1437" covered="0"/>
  <du var="entrySize" def="1426" use="1443" covered="0"/>
  <du var="entrySize" def="1426" use="1440" covered="0"/>
  <du var="handle" def="1428" use="1432" covered="0"/>
  <du var="success" def="1434" use="1438" target="1439" covered="0"/>
  <du var="success" def="1434" use="1438" target="1442" covered="0"/>
  <du var="elapsedNanos" def="1437" use="1442" covered="0"/>
  <du var="elapsedNanos" def="1437" use="1439" covered="0"/>
  <counter type="DU" missed="27" covered="0"/>
  <counter type="METHOD" missed="1" covered="0"/>
</method>
```


Snippet 19 Ba-Dua data v0 readEntry

```
<method name="readEntry" desc="(JJ)Lio/netty/buffer/ByteBuf;">
  <du var="this" def="1424" use="1433" covered="1"/>
  <du var="this" def="1424" use="1442" covered="0"/>
  <du var="this" def="1424" use="1443" covered="0"/>
  <du var="this" def="1424" use="1439" covered="1"/>
  <du var="this" def="1424" use="1440" covered="1"/>
  <du var="ledgerId" def="1424" use="1430" covered="0"/>
  <du var="entryId" def="1424" use="1432" covered="1"/>
  <du var="entryId" def="1424" use="1430" covered="0"/>
  <du var="LOG" def="1424" use="1429" target="1430" covered="0"/>
  <du var="LOG" def="1424" use="1429" target="1432" covered="1"/>
  <du var="LOG" def="1424" use="1430" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1433" covered="1"/>
  <du var="this.bookieStats" def="1424" use="1442" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1443" covered="0"/>
  <du var="this.bookieStats" def="1424" use="1439" covered="1"/>
  <du var="this.bookieStats" def="1424" use="1440" covered="1"/>
  <du var="NANOSECONDS" def="1424" use="1442" covered="0"/>
  <du var="NANOSECONDS" def="1424" use="1439" covered="1"/>
  <du var="requestNanos" def="1424" use="1437" covered="1"/>
  <du var="entrySize" def="1426" use="1443" covered="0"/>
  <du var="entrySize" def="1426" use="1440" covered="1"/>
  <du var="handle" def="1428" use="1432" covered="1"/>
  <du var="success" def="1434" use="1438" target="1439" covered="1"/>
  <du var="success" def="1434" use="1438" target="1442" covered="0"/>
  <du var="elapsedNanos" def="1437" use="1442" covered="0"/>
  <du var="elapsedNanos" def="1437" use="1439" covered="1"/>
  <counter type="DU" missed="12" covered="15"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Snippet 20 Ba-dua data per readEntry

```
▼<method name="addEntryForCompaction" desc="(JLio/netty/buffer/ByteBuf;)J">
   <du var="this" def="617" use="619" target="620" covered="1"/>
   <du var="this" def="617" use="619" target="623" covered="1"/>
   <du var="this" def="617" use="623" covered="1"/>
   <du var="this" def="617" use="626" covered="1"/>
   <du var="this" def="617" use="628" covered="1"/>
   <du var="this" def="617" use="629" covered="1"/>
   <du var="this" def="617" use="630" covered="1"/>
   <du var="this" def="617" use="631" covered="1"/>
   <du var="this" def="617" use="620" covered="1"/>
   <du var="ledgerId" def="617" use="630" covered="1"/>
   <du var="entry" def="617" use="625" covered="1"/>
   <du var="entry" def="617" use="629" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="619" target="620" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="619" target="623" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="626" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="628" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="629" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="630" covered="1"/>
   <du var="this.compactionLogChannel" def="617" use="631" covered="1"/>
   <du var="this.sizeBuffer" def="617" use="623" covered="1"/>
   <du var="entrySize" def="618" use="630" covered="1"/>
   <counter type="DU" missed="0" covered="22"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 21 Ba-Dua data per addEntryForCompaction

```
▼<method name="internalReadEntry" desc="(JJJZ)Lio/netty/buffer/ByteBuf;">
   <du var="this" def="818" use="839" covered="1"/>
   <du var="this" def="818" use="840" covered="1"/>
   <du var="this" def="818" use="830" covered="0"/>
   <du var="ledgerId" def="818" use="850" covered="0"/>
   <du var="ledgerId" def="818" use="830" covered="0"/>
   <du var="entryId" def="818" use="850" covered="0"/>
   <du var="entryId" def="818" use="830" covered="0"/>
   <du var="validateEntry" def="818" use="829" target="830" covered="0"/>
   <du var="validateEntry" def="818" use="829" target="837" covered="1"/>
   <du var="this.allocator" def="818" use="839" covered="1"/>
   <du var="entryLogId" def="818" use="840" covered="1"/>
   <du var="entryLogId" def="818" use="850" covered="0"/>
   <du var="entryLogId" def="818" use="830" covered="0"/>
   <du var="pos" def="819" use="840" covered="1"/>
   <du var="pos" def="819" use="850" covered="0"/>
   <du var="pos" def="819" use="830" covered="0"/>
   <du var="fc" def="825" use="840" covered="1"/>
   <du var="sizeBuff" def="827" use="830" covered="0"/>
   <du var="entrySize" def="828" use="839" covered="1"/>
   <du var="entrySize" def="828" use="841" target="849" covered="0"/>
   <du var="entrySize" def="828" use="841" target="854" covered="1"/>
   <du var="entrySize" def="828" use="854" covered="1"/>
   <du var="entrySize" def="828" use="850" covered="0"/>
   <du var="data" def="839" use="854" covered="1"/>
   <du var="data" def="839" use="856" covered="1"/>
   <du var="data" def="839" use="849" covered="0"/>
   <du var="rc" def="840" use="841" target="849" covered="0"/>
   <du var="rc" def="840" use="841" target="854" covered="1"/>
   <du var="rc" def="840" use="850" covered="0"/>
   <counter type="DU" missed="16" covered="13"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 22 Ba-Dua data v1 per internalReadEntry

```
▼<method name="internalReadEntry" desc="(JJJZ)Lio/netty/buffer/ByteBuf;">
   <du var="this" def="818" use="839" covered="1"/>
   <du var="this" def="818" use="840" covered="1"/>
   <du var="this" def="818" use="830" covered="1"/>
   <du var="ledgerId" def="818" use="850" covered="0"/>
   <du var="ledgerId" def="818" use="830" covered="1"/>
   <du var="entryId" def="818" use="850" covered="0"/>
   <du var="entryId" def="818" use="830" covered="1"/>
   <du var="validateEntry" def="818" use="829" target="830" covered="1"/>
   <du var="validateEntry" def="818" use="829" target="837" covered="1"/>
   <du var="this.allocator" def="818" use="839" covered="1"/>
   <du var="entryLogId" def="818" use="840" covered="1"/>
   <du var="entryLogId" def="818" use="850" covered="0"/>
   <du var="entryLogId" def="818" use="830" covered="1"/>
   <du var="pos" def="819" use="840" covered="1"/>
   <du var="pos" def="819" use="850" covered="0"/>
   <du var="pos" def="819" use="830" covered="1"/>
   <du var="fc" def="825" use="840" covered="1"/>
   <du var="sizeBuff" def="827" use="830" covered="1"/>
   <du var="entrySize" def="828" use="839" covered="1"/>
   <du var="entrySize" def="828" use="841" target="849" covered="0"/>
   <du var="entrySize" def="828" use="841" target="854" covered="1"/>
   <du var="entrySize" def="828" use="854" covered="1"/>
   <du var="entrySize" def="828" use="850" covered="0"/>
   <du var="data" def="839" use="854" covered="1"/>
   <du var="data" def="839" use="856" covered="1"/>
   <du var="data" def="839" use="849" covered="0"/>
   <du var="rc" def="840" use="841" target="849" covered="0"/>
   <du var="rc" def="840" use="841" target="854" covered="1"/>
   <du var="rc" def="840" use="850" covered="0"/>
   <counter type="DU" missed="9" covered="20"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 23 Ba-Dua data v2 per internalReadEntry

```
▼<method name="executeQueryAndGetResult" desc="(Ljava/lang/String;)Ljava/sql/ResultSet;">
   <du var="this" def="194" use="203" covered="0"/>
   <du var="this" def="194" use="201" covered="1"/>
   <du var="this" def="194" use="205" covered="1"/>
   <du var="$SwitchMap$org$apache$tajo$ipc$ClientProtos$SubmitQueryResponse$ResultType" def="194" use="199" target="205" covered="1"/>
   <du var="$SwitchMap$org$apache$tajo$ipc$ClientProtos$SubmitQueryResponse$ResultType" def="194" use="199" target="201" covered="1"/>
   <du var="$SwitchMap$org$apache$tajo$ipc$ClientProtos$SubmitQueryResponse$ResultType" def="194" use="199" target="203" covered="0"/>
   <du var="this.defaultFetchRows" def="194" use="201" covered="1"/>
   <du var="response" def="194" use="199" target="205" covered="1"/>
   <du var="response" def="194" use="199" target="201" covered="1"/>
   <du var="response" def="194" use="199" target="203" covered="0"/>
   <du var="response" def="194" use="201" covered="1"/>
   <du var="queryId" def="197" use="203" covered="0"/>
   <du var="queryId" def="197" use="205" covered="1"/>
   <counter type="DU" missed="4" covered="9"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 24 Ba-dua data per executeQueryAndGetResult

```
<method name="executeQuery" desc="(Ljava/lang/String;)Lorg/apache/tajo/ipc/ClientProtos$SubmitQueryResponse;">
  <du var="this" def="161" use="172" covered="1"/>
  <du var="this.conn" def="161" use="172" covered="1"/>
  <du var="response" def="166" use="171" target="172" covered="1"/>
  <du var="response" def="166" use="171" target="175" covered="1"/>
  <du var="response" def="166" use="175" covered="1"/>
  <du var="response" def="166" use="172" covered="1"/>
  <counter type="DU" missed="0" covered="6"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Snippet 25 Ba-dua data per executeQuery

```
<method name="getFinishedQuery" desc="(Lorg/apache/tajo/QueryId;)Lorg/apache/tajo/master/QueryInfo;">
  <du var="this" def="175" use="179" covered="1"/>
  <du var="queryId" def="175" use="179" covered="1"/>
  <du var="this.masterContext" def="175" use="179" covered="1"/>
  <du var="queryInfo" def="176" use="178" target="179" covered="1"/>
  <du var="queryInfo" def="176" use="178" target="181" covered="1"/>
  <du var="queryInfo" def="176" use="181" covered="1"/>
  <du var="queryInfo" def="179" use="181" covered="1"/>
  <counter type="DU" missed="0" covered="7"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Snippet 26 Ba-Dua data per getFinishedQuery

```
<method name="getQueryInProgress" desc="(Lorg/apache/tajo/QueryId;)Lorg/apache/tajo/master/QueryInProgress;">
  <du var="this" def="269" use="272" covered="1"/>
  <du var="queryId" def="269" use="272" covered="1"/>
  <du var="this.runningQueries" def="269" use="272" covered="1"/>
  <du var="queryInProgress" def="269" use="271" target="272" covered="1"/>
  <du var="queryInProgress" def="269" use="271" target="275" covered="0"/>
  <du var="queryInProgress" def="269" use="275" covered="0"/>
  <du var="queryInProgress" def="272" use="275" covered="1"/>
  <counter type="DU" missed="2" covered="5"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Snippet 27 Ba-dua per getQueryInProgress

```
<method name="getQueryInProgress" desc="(Lorg/apache/tajo/QueryId;)Lorg/apache/tajo/master/QueryInProgress;">
  <du var="this" def="269" use="272" covered="1"/>
  <du var="queryId" def="269" use="272" covered="1"/>
  <du var="this.runningQueries" def="269" use="272" covered="1"/>
  <du var="queryInProgress" def="269" use="271" target="272" covered="1"/>
  <du var="queryInProgress" def="269" use="271" target="275" covered="1"/>
  <du var="queryInProgress" def="269" use="275" covered="1"/>
  <du var="queryInProgress" def="272" use="275" covered="1"/>
  <counter type="DU" missed="0" covered="7"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Snippet 28 Ba-dua data v2 per getQueryInProgress

```
▼<method name="serviceInit" desc="(Lorg/apache/hadoop/conf/Configuration;)V">
   <du var="this" def="62" use="72" covered="1"/>
   <du var="this" def="62" use="72" covered="1"/>
   <du var="this" def="62" use="73" covered="1"/>
   <du var="this" def="62" use="75" covered="1"/>
   <du var="this" def="62" use="80" covered="1"/>
   <du var="this" def="62" use="81" covered="1"/>
   <du var="this" def="62" use="82" covered="1"/>
   <du var="WORKER_PEER_RPC_ADDRESS" def="62" use="81" covered="1"/>
   <du var="WORKER_RPC_SERVER_WORKER_THREAD_NUM" def="62" use="71" covered="1"/>
   <du var="LOG" def="62" use="80" covered="1"/>
   <du var="tajoConf" def="62" use="71" covered="1"/>
   <du var="tajoConf" def="62" use="81" covered="1"/>
   <du var="tajoConf" def="62" use="82" covered="1"/>
   <du var="initIsa" def="65" use="67" target="68" covered="0"/>
   <du var="initIsa" def="65" use="67" target="71" covered="1"/>
   <du var="initIsa" def="65" use="72" covered="1"/>
   <du var="initIsa" def="65" use="68" covered="0"/>
   <counter type="DU" missed="2" covered="15"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 29 Ba-dua data per serviceInit

```
▼<method name="stopQuery" desc="(Lorg/apache/tajo/QueryId;)V">
   <du var="this" def="279" use="284" covered="1"/>
   <du var="this" def="279" use="284" covered="1"/>
   <du var="this" def="279" use="285" covered="1"/>
   <du var="this" def="279" use="288" covered="1"/>
   <du var="this" def="279" use="289" covered="1"/>
   <du var="this" def="279" use="292" target="293" covered="1"/>
   <du var="this" def="279" use="292" target="296" covered="0"/>
   <du var="this" def="279" use="296" target="297" covered="0"/>
   <du var="this" def="279" use="296" target="300" covered="1"/>
   <du var="this" def="279" use="300" covered="1"/>
   <du var="this" def="279" use="304" covered="1"/>
   <du var="this" def="279" use="306" covered="1"/>
   <du var="this" def="279" use="302" covered="0"/>
   <du var="this" def="279" use="297" covered="0"/>
   <du var="this" def="279" use="293" covered="1"/>
   <du var="queryId" def="279" use="308" covered="1"/>
   <du var="queryId" def="279" use="288" covered="1"/>
   <du var="queryId" def="279" use="289" covered="1"/>
   <du var="LOG" def="279" use="308" covered="1"/>
   <du var="this.historyCache" def="279" use="284" covered="1"/>
   <du var="this.historyCache" def="279" use="284" covered="1"/>
   <du var="this.historyCache" def="279" use="285" covered="1"/>
   <du var="this.submittedQueries" def="279" use="288" covered="1"/>
   <du var="this.runningQueries" def="279" use="289" covered="1"/>
   <du var="this.minExecutionTime" def="279" use="292" target="293" covered="1"/>
   <du var="this.minExecutionTime" def="279" use="292" target="296" covered="0"/>
   <du var="this.minExecutionTime" def="279" use="293" covered="1"/>
   <du var="this.maxExecutionTime" def="279" use="296" target="297" covered="0"/>
   <du var="this.maxExecutionTime" def="279" use="296" target="300" covered="1"/>
   <du var="this.maxExecutionTime" def="279" use="297" covered="0"/>
   <du var="this.executedQuerySize" def="279" use="300" covered="1"/>
   <du var="this.executedQuerySize" def="279" use="306" covered="1"/>
   <du var="this.executedQuerySize" def="279" use="302" covered="0"/>
   <du var="this.avgExecutionTime" def="279" use="300" covered="1"/>
   <du var="this.avgExecutionTime" def="279" use="304" covered="1"/>
   <du var="this.avgExecutionTime" def="279" use="302" covered="0"/>
   <du var="queryInProgress" def="280" use="281" target="282" covered="1"/>
   <du var="queryInProgress" def="280" use="281" target="308" covered="1"/>
   <du var="queryInProgress" def="280" use="282" covered="1"/>
   <du var="queryInProgress" def="280" use="283" covered="1"/>
   <du var="executionTime" def="291" use="292" target="293" covered="1"/>
   <du var="executionTime" def="291" use="292" target="296" covered="0"/>
   <du var="executionTime" def="291" use="296" target="297" covered="0"/>
   <du var="executionTime" def="291" use="296" target="300" covered="1"/>
   <du var="executionTime" def="291" use="304" covered="1"/>
   <du var="executionTime" def="291" use="302" covered="0"/>
   <du var="executionTime" def="291" use="297" covered="0"/>
   <du var="executionTime" def="291" use="293" covered="1"/>
   <du var="totalExecutionTime" def="300" use="301" target="302" covered="0"/>
   <du var="totalExecutionTime" def="300" use="301" target="304" covered="1"/>
   <du var="totalExecutionTime" def="300" use="302" covered="0"/>
   <counter type="DU" missed="15" covered="36"/>
   <counter type="METHOD" missed="0" covered="1"/>
 </method>
```

Snippet 30 Ba-dua data per stopQuery

```
1409        /**
1410         * Fences a ledger. From this point on, clients will be unable to
1411         * write to this ledger. Only recoveryAddEntry will be
1412         * able to add entries to the ledger.
1413         * This method is idempotent. Once a ledger is fenced, it can
1414         * never be unfenced. Fencing a fenced ledger has no effect.
1415         */
1416        public SettableFuture<Boolean> fenceLedger(long ledgerId, byte[] masterKey)
1417                throws IOException, BookieException {
1418            LedgerDescriptor handle = handles.getHandle(ledgerId, masterKey);
1419 1          return handle.fenceAndLogInJournal(getJournal(ledgerId));
1420        }
```

Snippet 31 Pit data per fenceLedger

```
638     public static LedgerStorage mountLedgerStorageOffline(ServerConfiguration conf, LedgerStorage ledgerStorage)
639             throws IOException {
640         StatsLogger statsLogger = NullStatsLogger.INSTANCE;
641         DiskChecker diskChecker = new DiskChecker(conf.getDiskUsageThreshold(), conf.getDiskUsageWarnThreshold());
642
643         LedgerDirsManager ledgerDirsManager = createLedgerDirsManager(
644                 conf, diskChecker, statsLogger.scope(LD_LEDGER_SCOPE));
645         LedgerDirsManager indexDirsManager = createIndexDirsManager(
646                 conf, diskChecker, statsLogger.scope(LD_INDEX_SCOPE), ledgerDirsManager);
647
648 1       if (null == ledgerStorage) {
649             ledgerStorage = buildLedgerStorage(conf);
650         }
651
652         CheckpointSource checkpointSource = new CheckpointSource() {
653             @Override
654             public Checkpoint newCheckpoint() {
655                 return Checkpoint.MAX;
656             }
657
658             @Override
659             public void checkpointComplete(Checkpoint checkpoint, boolean compact)
660                     throws IOException {
661             }
662         };
663
664         Checkpointer checkpointer = Checkpointer.NULL;
665
666 1       ledgerStorage.initialize(
667                 conf,
668                 null,
669                 ledgerDirsManager,
670                 indexDirsManager,
671                 null,
672                 checkpointSource,
673                 checkpointer,
674                 statsLogger,
675                 UnpooledByteBufAllocator.DEFAULT);
676
677 1       return ledgerStorage;
678     }
```

Snippet 32 Pit data per per mountLedgerStorageOffline

```
1416      public SettableFuture<Boolean> fenceLedger(long ledgerId, byte[] masterKey)
1417              throws IOException, BookieException {
1418          LedgerDescriptor handle = handles.getHandle(ledgerId, masterKey);
1419 1        return handle.fenceAndLogInJournal(getJournal(ledgerId));
1420      }
1421
1422      public ByteBuf readEntry(long ledgerId, long entryId)
1423              throws IOException, NoLedgerException {
1424          long requestNanos = MathUtils.nowInNano();
1425          boolean success = false;
1426          int entrySize = 0;
1427          try {
1428              LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
1429              if (LOG.isTraceEnabled()) {
1430                  LOG.trace("Reading {}@{}", entryId, ledgerId);
1431              }
1432              ByteBuf entry = handle.readEntry(entryId);
1433 1            bookieStats.getReadBytes().add(entry.readableBytes());
1434              success = true;
1435 1            return entry;
1436          } finally {
1437              long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
1438 1            if (success) {
1439 1                bookieStats.getReadEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1440 1                bookieStats.getReadBytesStats().registerSuccessfulValue(entrySize);
1441              } else {
1442 1                bookieStats.getReadEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
1443 1                bookieStats.getReadEntryStats().registerFailedValue(entrySize);
1444              }
1445          }
1446      }
```

Snippet 33 Pit data per readEntry

```
616       long addEntryForCompaction(long ledgerId, ByteBuf entry) throws IOException {
617           synchronized (compactionLogLock) {
618 1             int entrySize = entry.readableBytes() + 4;
619 1             if (compactionLogChannel == null) {
620 1                 createNewCompactionLog();
621               }
622
623               ByteBuf sizeBuffer = this.sizeBuffer.get();
624               sizeBuffer.clear();
625               sizeBuffer.writeInt(entry.readableBytes());
626 1             compactionLogChannel.write(sizeBuffer);
627
628               long pos = compactionLogChannel.position();
629 1             compactionLogChannel.write(entry);
630 1             compactionLogChannel.registerWrittenEntry(ledgerId, entrySize);
631 3             return (compactionLogChannel.getLogId() << 32L) | pos;
632           }
633       }
```

Snippet 34 Pit data per addEntryForCompaction

```
816     public ByteBuf internalReadEntry(long ledgerId, long entryId, long location, boolean validateEntry)
817             throws IOException, Bookie.NoEntryException {
818         long entryLogId = logIdForOffset(location);
819         long pos = posForOffset(location);
820
821
822         BufferedReadChannel fc = null;
823         int entrySize = -1;
824         try {
825             fc = getFCForEntryInternal(ledgerId, entryId, entryLogId, pos);
826
827             ByteBuf sizeBuff = readEntrySize(ledgerId, entryId, entryLogId, pos, fc);
828             entrySize = sizeBuff.getInt(0);
829 1           if (validateEntry) {
830 1               validateEntry(ledgerId, entryId, entryLogId, pos, sizeBuff);
831             }
832         } catch (EntryLookupException.MissingEntryException entryLookupError) {
833             throw new Bookie.NoEntryException("Short read from entrylog " + entryLogId,
834                     ledgerId, entryId);
835         } catch (EntryLookupException e) {
836             throw new IOException(e.toString());
837         }
838
839         ByteBuf data = allocator.buffer(entrySize, entrySize);
840         int rc = readFromLogChannel(entryLogId, fc, data, pos);
841 1       if (rc != entrySize) {
842             // Note that throwing NoEntryException here instead of IOException is not
843             // without risk. If all bookies in a quorum throw this same exception
844             // the client will assume that it has reached the end of the ledger.
845             // However, this may not be the case, as a very specific error condition
846             // could have occurred, where the length of the entry was corrupted on all
847             // replicas. However, the chance of this happening is very very low, so
848             // returning NoEntryException is mostly safe.
849             data.release();
850             throw new Bookie.NoEntryException("Short read for " + ledgerId + "@"
851                                               + entryId + " in " + entryLogId + "@"
852                                               + pos + "(" + rc + "!=" + entrySize + ")", ledgerId, entryId);
853         }
854         data.writerIndex(entrySize);
855
856 1       return data;
857     }
```

Snippet 35 Pit data per internalReadEntry

```
638     public static LedgerStorage mountLedgerStorageOffline(ServerConfiguration conf, LedgerStorage ledgerStorage)
639             throws IOException {
640         StatsLogger statsLogger = NullStatsLogger.INSTANCE;
641         DiskChecker diskChecker = new DiskChecker(conf.getDiskUsageThreshold(), conf.getDiskUsageWarnThreshold());
642
643         LedgerDirsManager ledgerDirsManager = createLedgerDirsManager(
644                 conf, diskChecker, statsLogger.scope(LD_LEDGER_SCOPE));
645         LedgerDirsManager indexDirsManager = createIndexDirsManager(
646                 conf, diskChecker, statsLogger.scope(LD_INDEX_SCOPE), ledgerDirsManager);
647
648 1       if (null == ledgerStorage) {
649             ledgerStorage = buildLedgerStorage(conf);
650         }
651
652         CheckpointSource checkpointSource = new CheckpointSource() {
653             @Override
654             public Checkpoint newCheckpoint() {
655                 return Checkpoint.MAX;
656             }
657
658             @Override
659             public void checkpointComplete(Checkpoint checkpoint, boolean compact)
660                     throws IOException {
661             }
662         };
663
664         Checkpointer checkpointer = Checkpointer.NULL;
665
666 1       ledgerStorage.initialize(
667                 conf,
668                 null,
669                 ledgerDirsManager,
670                 indexDirsManager,
671                 null,
672                 checkpointSource,
673                 checkpointer,
674                 statsLogger,
675                 UnpooledByteBufAllocator.DEFAULT);
676
677 1       return ledgerStorage;
678     }
```

Snippet 36 Pit data v2 per mountLedgerStorageOffline

```
158     @Override
159     public SubmitQueryResponse executeQuery(final String sql) {
160
161       final BlockingInterface stub = conn.getTMStub();
162       final QueryRequest request = buildQueryRequest(sql, false);
163
164       SubmitQueryResponse response;
165       try {
166         response = stub.submitQuery(null, request);
167       } catch (ServiceException e) {
168         throw new RuntimeException(e);
169       }
170
171       if (isSuccess(response.getState())) {
172         conn.updateSessionVarsCache(ProtoUtil.convertToMap(response.getSessionVars()));
173       }
174
175 1     return response;
176
```

Snippet 37 Pit data per executeQuery

```
191    @Override
192    public ResultSet executeQueryAndGetResult(String sql) throws TajoException {
193
194      SubmitQueryResponse response = executeQuery(sql);
195      throwIfError(response.getState());
196
197      QueryId queryId = new QueryId(response.getQueryId());
198
199      switch (response.getResultType()) {
200        case ENCLOSED:
201 1        return TajoClientUtil.createResultSet(this, response, defaultFetchRows);
202        case FETCH:
203 1        return this.getQueryResultAndWait(queryId);
204        default:
205 1        return this.createNullResultSet(queryId);
206      }
207    }
```

Snippet 38 Pit data per execeuteQueryAndGetResult

```
270    @Override
271    public QueryStatus getQueryStatus(QueryId queryId) throws QueryNotFoundException {
272
273      final BlockingInterface stub = conn.getTMStub();
274      final GetQueryStatusRequest request = GetQueryStatusRequest.newBuilder()
275          .setSessionId(conn.sessionId)
276          .setQueryId(queryId.getProto())
277          .build();
278
279      GetQueryStatusResponse res;
280      try {
281        res = stub.getQueryStatus(null, request);
282      } catch (ServiceException t) {
283        throw new RuntimeException(t);
284      }
285
286      throwsIfThisError(res.getState(), QueryNotFoundException.class);
287      ensureOk(res.getState());
288 1    return new QueryStatus(res);
289    }
```

Snippet 39 Pit data per getQueryStatus

```
172    public QueryInfo getFinishedQuery(QueryId queryId) {
173      try {
174        QueryInfo queryInfo;
175        synchronized (historyCache) {
176          queryInfo = (QueryInfo) historyCache.get(queryId);
177        }
178        if (queryInfo == null) {
179          queryInfo = this.masterContext.getHistoryReader().getQueryByQueryId(queryId);
180        }
181 1      return queryInfo;
182      } catch (Throwable e) {
183        LOG.error(e.getMessage(), e);
184 1      return null;
185      }
186    }
```

Snippet 40 Pit data per getFinishedQuery

```
267     public QueryInProgress getQueryInProgress(QueryId queryId) {
268       QueryInProgress queryInProgress;
269       queryInProgress = submittedQueries.get(queryId);
270
271       if (queryInProgress == null) {
272         queryInProgress = runningQueries.get(queryId);
273       }
274
275 1     return queryInProgress;
276     }
```

Snippet 41Pit data per QueryInProgress

```
79    @Override
80    public void serviceInit(Configuration conf) throws Exception {
81      try {
82        this.dispatcher = new AsyncDispatcher();
83        addService(this.dispatcher);
84
85        this.dispatcher.register(QueryJobEvent.Type.class, new QueryJobManagerEventHandler());
86
87        TajoConf tajoConf = TUtil.checkTypeAndGet(conf, TajoConf.class);
88        this.historyCache = new LRUMap(tajoConf.getIntVar(TajoConf.ConfVars.HISTORY_QUERY_CACHE_SIZE));
89      } catch (Exception e) {
90        LOG.error("Failed to init service " + getName() + " by exception " + e, e);
91      }
92
93      super.serviceInit(conf);
94    }
```

Snippet 42 Pit data per serviceInit

```
278   public void stopQuery(QueryId queryId) {
279     LOG.info("Stop QueryInProgress:" + queryId);
280     QueryInProgress queryInProgress = getQueryInProgress(queryId);
281     if(queryInProgress != null) {
282       queryInProgress.stopProgress();
283       QueryInfo queryInfo = queryInProgress.getQueryInfo();
284       synchronized (historyCache) {
285         historyCache.put(queryInfo.getQueryId(), queryInfo);
286       }
287
288       submittedQueries.remove(queryId);
289       runningQueries.remove(queryId);
290
291       long executionTime = queryInfo.getFinishTime() - queryInfo.getStartTime();
292       if (executionTime < minExecutionTime.get()) {
293         minExecutionTime.set(executionTime);
294       }
295
296       if (executionTime > maxExecutionTime.get()) {
297         maxExecutionTime.set(executionTime);
298       }
299
300       long totalExecutionTime = executedQuerySize.get() * avgExecutionTime.get();
301       if (totalExecutionTime > 0) {
302         avgExecutionTime.set((totalExecutionTime + executionTime) / (executedQuerySize.get() + 1));
303       } else {
304         avgExecutionTime.set(executionTime);
305       }
306       executedQuerySize.incrementAndGet();
307     } else {
308       LOG.warn("No QueryInProgress while query stopping: " + queryId);
309     }
310   }
```

Snippet 43 Pit data per StopQuery