

# Report Modulo Software Testing

## Tiziano Taglienti - 0304926

### 1. Introduzione

L'obiettivo di questo report è presentare i risultati dell'attività di testing su classi Java di progetti open source di Apache.

Per lo sviluppo dei test è stato configurato un ambiente di lavoro a partire dal fork su GitHub (Git come software di Version Control) dei due progetti selezionati, cioè Apache Bookkeeper e Apache Storm, e poi impostato in entrambi i casi a una versione precedente, per questioni di compatibilità con i programmi installati necessari per lo svolgimento del progetto.

A supporto del processo di testing è stato utilizzato TravisCI per la build del progetto su una macchina virtuale, perché per il processo di continuous integration è importante che le fasi di building e testing siano automatizzate ogni volta che viene effettuata una modifica al progetto, che viene rilevata attraverso i commit in Git.

Inoltre sono stati utilizzati: Maven come sistema di build, per la gestione delle librerie e l'esecuzione dei test; JUnit per l'esecuzione di questi ultimi; SonarCloud per analizzare la coverage dei test in remoto e JaCoCo per fare la stessa cosa in locale; Ba-dua per l'analisi della data flow coverage e Pit per il mutation testing.

La scelta delle classi deriva dall'analisi della documentazione ove possibile, maggiormente per Bookkeeper. Nel processi di sviluppo dei casi di test è stato adottato un approccio principalmente unidimensionale, in cui ciascun parametro è stato esaminato in modo isolato dagli altri, garantendo così un'analisi indipendente.

Di conseguenza, tale metodologia ha permesso di progettare dei casi di test in grado di coprire in maniera esaustiva tutte le classi di equivalenza definite, per garantire una copertura completa delle possibili situazioni e configurazioni.

Nella maggior parte dei casi, i test subiscono una prima fase di partizionamento dei parametri ad alto livello, per poi essere approfonditi, con lo scopo di aumentare la qualità del test evidenziando le differenze tra le fasi. Infine va detto che la maggior parte del tempo è stato impiegato per la risoluzione dei numerosi problemi riscontrati nell'esecuzione di Maven, piuttosto che dalla scrittura dei test.

### 2. Bookkeeper

#### 2.1. Presentazione

Apache Bookkeeper è un progetto open source sviluppato da Apache Software Foundation.

Si tratta di un sistema di archiviazione distribuita che si concentra sulla gestione di log affidabili e altamente disponibili, caratterizzato da tolleranza ai guasti, scalabilità e tempi di latenza minimi.

Il suo utilizzo principale è nell'ambito di applicazioni e servizi che richiedono una registrazione affidabile e sequenziale delle attività, come per esempio i sistemi di messaggistica, i database distribuiti e i servizi di streaming.

#### 2.2. Scelta delle classi e dei metodi

Nel contesto di questo progetto sono state individuate le seguenti classi di test: ReadCache e WriteCache all'interno del package `bookie.storage.ldb` e ByteBufList nel package `util`, tutte all'interno del modulo `bookkeeper-server`.

La scelta delle classi ReadCache e WriteCache è basata principalmente sulla chiarezza che è stata riscontrata nella loro documentazione, piuttosto che su uno studio delle metriche.

La scelta di ByteBufList si deve alla numerosità di commit subiti dalla classe, seppure sia una classe abbastanza recente nella versione considerata. Proprio questa peculiarità l'ha resa un buon candidato per la ricerca di bug, giustificando la sua inclusione nei casi di test.

#### 2.3. ByteBufList

Questa classe agisce come un gestore per una sequenza di oggetti ByteBuf, i quali sono componenti della libreria Java e presentano le funzionalità di una coda FIFO di byte, con le posizioni di testa e coda.

Quando viene scritto un dato nel ByteBuf, la coda si estende di una quantità pari al numero di byte corrispondente. Allo stesso modo, durante la lettura di un dato, la posizione della testa aumenta del numero di byte letti.

La classe in questione non cerca di imitare la classe ByteBuf, ma si espone piuttosto come un oggetto destinato a essere codificato su un canale.

Esistono due tipi di codificatori: un *encoder* che scriverà tutti i buffer nella ByteBufList attraverso il canale e un *encoder\_with\_size*, simile al precedente ma con l'aggiunta di 4 byte iniziali che fungono da intestazione, contenenti il numero di byte che possono essere letti attraverso tutti i buffer presenti nella ByteBufList.

La classe di test è composta da una fase di setup in cui si crea una cartella temporanea in cui verrà generato un file vuoto da usare nelle fasi successive, e dai metodi *testGetBytesAndArrays* e *testEncoderWithWrite*, che testano i metodi:

- *public static ByteBufList get(ByteBuf b1, ByteBuf b2)*: si crea una ByteBufList inizializzata con due buffer che vengono gestiti dall'istanza della lista;
- *public void prepend(ByteBuf buf)*: si aggiunge un oggetto ByteBuf all'inizio della lista;
- *public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise)*: metodo contenuto nella classe interna *Encoder*, progettato per scrivere oggetti sul canale in modo asincrono.

*testGetBytesAndArrays* verifica che l'istanza creata dall'oggetto ByteBufList, contenente due buffer composti da due oggetti di tipo String, gestisca correttamente l'aggiunta di buffer di byte, la concatenazione di dati e la copia dei byte in un array.

*testEncoderWithWrite* scrive il contenuto di un buffer in un file e verifica che tale scrittura sia stata eseguita leggendo il file.

### 2.3.1. Individuazione dominio di input e partizionamento dominio

- *public static ByteBufList get(...)*: durante il test del metodo, i parametri di input sono rispettivamente *Unpooled.wrappedBuffer(this.stringPart1.getBytes())* e *Unpooled.wrappedBuffer(this.stringPart2.getBytes())*, e le partizioni scelte inizialmente sono:

```
stringPart1 = {stringa vuota, stringa non vuota}  
stringPart2 = {stringa vuota, stringa non vuota}
```

Entrando nel particolare, i possibili valori che poi vengono assegnati alle classi di equivalenza sono:

```
stringPart1 = {"test", " "}  
stringPart2 = {"part", " "}
```

e i valori vengono combinati inizialmente seguendo un approccio unidimensionale, quindi stabilendo che ognuno dei due valori compaia almeno una volta in un caso di test.

[[Casi di test](#)] - Questi semplici casi di test creati sono abbastanza per raggiungere da subito una statement coverage del 100% per il metodo *get* ([figura 1](#)), dopo aver testato che la dimensione in byte di *stringPart1 + stringPart2* sia uguale al risultato ottenuto chiamando il metodo *toArray()* sulla lista.

Il test per questo metodo è già accettabile senza bisogno di coprire altri scenari.

- *public void prepend(...)*: i valori assegnati al parametro di input dipendono dalla partizione:

```
prefix = {stringa vuota, stringa non vuota}
```

In particolare, i valori che assegnati alle classi di equivalenza per *prefix* sono {"pre", " "}.

Anche in questo caso, questi valori sono abbastanza per garantire una statement coverage massima ([figura 1](#)). Il test a seguito dell'operazione di *prepend* consiste nella verifica che la dimensione in byte di *prefix + stringPart1 + stringPart2* sia uguale al risultato di *toArray()* applicato alla lista.

Alla fine di *testGetBytesAndArrays* viene eseguito un controllo sulla lunghezza di *prefix + stringPart1 + stringPart2*, che sia uguale al numero di byte copiati dal metodo *getBytes* (anche questo con coverage massima) applicato sull'array con dimensione pari al valore di *this.numBytes*, che viene prima partizionato in {<=0, >0} e poi assume i valori in questa classe di equivalenza:

```
numBytes = {0, 1, 11},
```

scelti in questo modo a seguito di un'analisi boundary values, a cui è stato aggiunto un valore più lontano dallo zero.

- `public void write(...)`: per questo metodo le partizioni iniziali dei parametri sono determinate dalle condizioni e dal contesto in cui vengono usati durante il test:

```
ctx = mock(ChannelHandlerContext.class)
msg = {stringa, oggetto ByteBufList}
promise = null
```

Il parametro `ctx` è un mock di `ChannelHandlerContext` e rappresenta un oggetto simulato che fornisce un ambiente controllato per il test; `msg` è l'oggetto da scrivere, per cui vengono definite le classi di equivalenza:

```
msg = {"testString", ByteBufList.get(Unpooled.wrappedBuffer("testString".getBytes())),
ByteBufList.get(Unpooled.wrappedBuffer("testStringPart1".getBytes()))}
```

mentre `promise` assume il valore null all'interno del test perché non è necessario al fine della scrittura del file.

Oltre a questi parametri, nel metodo `write` riveste un ruolo chiave il parametro `encoder`, che determina in che modo i dati debbano essere scritti sul canale. L'encoder decide come trattare l'oggetto `msg` prima di scriverlo.

In un primo momento a questo parametro viene assegnato solo il valore `ByteBufList.ENCODER`.

Il metodo ha inizialmente una statement coverage del 74% e una branch coverage dell'87% ([figura 2](#)) a causa della mancata copertura del caso in cui `prependSize` è true ([figura 3](#)); invece la data flow coverage in questo caso è al 79% ([figura 4](#)).

A questa situazione iniziale sono state introdotte delle modifiche, come l'aggiunta del valore `ByteBufList.ENCODER_WITH_SIZE` all'encoder per coprire l'unico branch non coperto, per quando `prependSize` è true.

Inoltre, è stato aggiunto un nuovo comportamento all'oggetto `ctx`, per cui quando viene chiamato `ctx.alloc()` viene stanziato l'oggetto `ByteBuffAllocator` richiesto dal metodo `directBuffer(int, int)`.

Questi miglioramenti conducono al raggiungimento della coverage massima per il metodo `write` ([figura 5](#)), e anche la data flow coverage aumenta e arriva al 94% ([figura 6](#)). In quest'ultimo caso non si raggiunge il 100% a causa di una variabile locale definita per un ciclo for e che non può essere coperta. Per quanto riguarda la mutation coverage del metodo, nella situazione iniziale era al 48% ([figura 7](#)). Attraverso le modifiche non è stato possibile aumentarla, infatti risulta essere al 46% ([figura 8](#)).

## 2.4. ReadCache

Questa classe è responsabile dell'implementazione di una cache di lettura, che opera utilizzando una quantità di memoria specificata e la associa a una struttura dati di tipo hashmap.

Questa cache di lettura è suddivisa in più segmenti che vengono utilizzati in modo circolare, seguendo un approccio "ring-buffer". Quando la cache raggiunge la sua capacità massima, il segmento più vecchio viene cancellato e riutilizzato per fare spazio alle nuove voci da aggiungere.

Il metodo testato è:

- `public ByteBuf get(long ledgerId, long entryId)`: consente di accedere a un buffer contenente l'entry all'interno della cache di lettura, identificata da un `entryId` specifico che deve essere letto dal ledger specificato da `ledgerId`.

Nel caso in cui l'entry non sia presente in nessun segmento della cache, il metodo restituirà null.

### 2.4.1. Individuazione dominio di input e partizionamento dominio

Per testare l'unico metodo della classe è stata creata un'istanza di `ReadCache` nella quale sono state inserite delle entry tramite il metodo `ReadCache.put()`.

Sono stati definiti separatamente gli identificatori dei ledger e della entry da inserire nella cache rispetto a quelli utilizzati nel metodo `get()`.

Analizzando i parametri di input, avendo solo due interi, il partizionamento iniziale è di questo tipo:

```
ledgerId = {< 0, ≥ 0}
entryId = {< 0, ≥ 0}
```

Per gli interi, la suddivisione in valori positivi e negativi serve a rappresentare i casi limite, poiché identificatori negativi rappresentano casi non validi e quelli positivi rappresentano casi regolari.

L'analisi dietro la scelta dei parametri per il metodo riguarda la separazione tra gli `idGet` e gli `idPut`.

Innanzitutto, i valori assegnati alle classi di equivalenza, scelti a seguito di un'analisi boundary values, sono:

$$\begin{aligned} ledgerId &= \{-1, 0, 1\} \\ entryId &= \{-1, 0, 1\} \end{aligned}$$

E nella scelta dei casi di test, le combinazioni sono selezionate anche osservando il rapporto che c'è tra *idGet* e *idPut*. Logicamente il put viene prima del get, quindi prima si scelgono i valori da assegnare a *ledgerIdPut* e *entryIdPut*, da cui dipendono i valori di *ledgerIdGet* e *entryIdGet*, che quindi appartengono a classi di equivalenza che distinguono i casi in cui

$$\{idGet \neq idPut, idGet == idPut\}$$

A seguito di questo ragionamento otteniamo i casi di test.

[Casi di test] - Il primo caso ritornerà false per un fail della put di un ledger con identificatore negativo; il secondo ritornerà false perché viene fatto il put di (1, 0) e il get di (0, 1), quindi è il caso in cui *idGet*  $\neq$  *idPut*, per entrambi gli id; il terzo caso è l'unico che non solleva eccezioni, riscontrando un successo del metodo *get()*.

Per quanto riguarda l'analisi della coverage, già con questi primi casi di test vengono raggiunte statement coverage e branch coverage del 100% ([figura 9](#), [figura 10](#)), e una data flow coverage del 97% ([figura 11](#)).

La mutation coverage della classe *ReadCache* è del 18% ([figura 12](#)) e viene mostrato il report di Pit per il metodo testato *get* ([figura 13](#)).

## 2.5. WriteCache

Questa classe implementa una cache di scrittura, che alloca la dimensione richiesta della memoria diretta e la suddivide in più segmenti. Le entry vengono aggiunte in un buffer comune e indicizzate attraverso una hashmap, finché la cache non viene cancellata.

I metodi testati sono:

- *public boolean put(long ledgerId, long entryId, ByteBuf entry)*: scrive una entry, identificata dal suo *entryId*, all'interno di un *ledger*, anch'esso identificato da un *ledgerId*.  
Questo metodo restituisce true se la scrittura ha successo e false in caso di fallimento.
- *public ByteBuf get(long ledgerId, long entryId)*: metodo duale del precedente, che va a prelevare una entry dalla *WriteCache*.  
In particolare, se i parametri forniti identificano una entry valida, il metodo restituisce un oggetto *ByteBuf* con il contenuto della entry, altrimenti viene restituito null.

Per condurre i test si separa l'uso di *ledgerId* e *entryId* da inserire nella cache attraverso il metodo *put* da quelli utilizzati nel metodo *get*.

### 2.5.1. Individuazione dominio di input e partizionamento dominio

- *public boolean put(...)*: analizzando i parametri di input, avendo due interi e un parametro di tipo *ByteBuf*, le partizioni iniziali sono state:

$$\begin{aligned} ledgerId &= \{< 0, \geq 0\} \\ entryId &= \{< 0, \geq 0\} \\ entry &= \{\text{entry size} \leq \text{available memory in cache}, \text{entry size} > \text{available memory in cache}\} \end{aligned}$$

Per gli interi, la divisione in valori positivi e negativi è utile per rappresentare casi limite e per garantire una copertura completa e significativa del comportamento del SUT in questo caso, dal momento che identificatori negativi rappresentano casi non validi, che si prevede che il metodo gestirà correttamente.

Inoltre, nella fase di setup si inizializza una *WriteCache*, dove il valore assunto da *maxSegmentSize* dipende dalla variabile booleana *notAvailableSegment*, inclusa nella lista dei parametri.

A partire dalle partizioni, i possibili valori delle classi di equivalenza sono:

$$\begin{aligned} ledgerId &= \{-1, 0, 1\} \\ entryId &= \{-1, 0, 1\} \\ entry &= \{\text{UnpooledByteBufAllocator.DEFAULT.buffer(1024)}, \\ &\quad \text{UnpooledByteBufAllocator.DEFAULT.buffer(11*1024)}\} \end{aligned}$$

I valori di *ledgerId* ed *entryId* vengono scelti a seguito di un'analisi boundary values che individua i valori rappresentativi della partizione, mentre per *entry* il discorso è diverso: il successo o il fallimento del metodo dipendono dalla quantità di byte da scrivere rispetto alla memoria disponibile nella cache,

quindi è importante suddividere i casi in cui questa quantità è inferiore o superiore alla capacità della cache.

Inizialmente i valori dei parametri vengono combinati seguendo un approccio unidimensionale, per semplicità.

[[Casi di test](#)] - Dal primo e dal terzo caso ci si aspetta un fallimento dell'operazione: nel primo caso perché il `ledgerId` negativo solleva una `IllegalArgumentException`, nel terzo caso perché la dimensione di `entry` è maggiore della dimensione della cache.

Il secondo caso conduce a un successo per il metodo `put`.

Ora, analizzando la coverage, si nota che il metodo `put` ha una statement coverage del 96% e una branch coverage del 62% ([figura 14](#)), e guardando ciò che non è stato testato ([figura 15](#)) si deduce che questi valori siano migliorabili aggiungendo dei casi di test.

Per quanto riguarda la data flow coverage, questa è all'80% ([figura 16](#)).

Viene riportata anche la mutation coverage del metodo `put` ([figura 17](#)).

Per provare ad aumentare la coverage del metodo `put`, una modifica utile può essere quella di considerare il valore null per `entry`; inoltre, adesso si vanno a creare nuovi casi orientandosi verso un approccio multidimensionale, ma senza arrivare al prodotto cartesiano tra tutti i parametri per non raggiungere dimensioni spropositate.

$\text{entry} = \{\text{null}, \text{entry size } \leq \text{available memory in cache}, \text{entry size } > \text{available memory in cache}\}$

In base a quanto detto, vengono introdotti ulteriori casi di test.

[[Casi di test](#)] - Oltre ad aggiungere il valore null per `entry`, ne vengono provati altri più vicini al boundary. Dal caso 6 ci si aspetta un successo dell'operazione di `put`, mentre gli altri tre casi falliranno, rispettivamente per una `NullPointerException` nel quarto caso e perché la dimensione di `entry` supera la dimensione della cache nei casi 5 e 7.

Dopo questi miglioramenti, si raggiunge una statement coverage del 97% e una branch coverage del 75% ([figura 18](#)), oltre alle quali non si riesce ad andare. La differenza sta nella riuscita copertura della condizione che prima non era coperta ([figura 19](#)).

La data flow coverage ora è al'87,5% ([figura 20](#)), e la mutation coverage aumenta di poco.

- `public ByteBuf get(long ledgerId, long entryId)`: analizzando i parametri di input, avendo solo due interi, il partizionamento iniziale viene fatto in questo modo:

$\text{ledgerId} = \{-1, 0, 1\}$   
 $\text{entryId} = \{-1, 0, 1\}$

Per entrambi i parametri viene usato un approccio con boundary values per individuare i valori rappresentativi per le partizioni, quindi i possibili valori per le classi di equivalenza sono:

$\text{ledgerId} = \{-1, 0, 1\}$   
 $\text{entryId} = \{-1, 0, 1\}$

[[Casi di test](#)] - I casi di test sono stati scelti, come per il `get` di `ReadCache`, distinguendo `idGet` e `idPut`. I quattro parametri in ordine rappresentano `ledgerIdGet`, `entryIdGet`, `ledgerIdPut` e `entryIdPut`, quindi il test si approfondisce testando i due casi `{idGet != idPut, idGet == idPut}`.

Il primo caso di test è l'unico che non solleva eccezioni e porta a un successo del metodo `get`.

Il secondo conduce a un fallimento a causa dell'operazione di `put` di un valore negativo e il terzo porta anch'esso a un fallimento perché `idGet` e `idPut` sono diversi per il ledger e per la entry.

Tra i parametri di `WriteCachePutTest` e `WriteCacheGetTest`, il primo è sempre `expectedResult` di tipo `boolean`, che indica appunto il risultato che ci si aspetta dal test, ossia un successo (true) o un fallimento (false), e il suo valore è in linea con ciò che viene detto nei commenti dei casi di test.

Questo parametro è oggetto del test tramite l'operazione di `assertEquals` con il risultato di `get`.

Per questo metodo viene raggiunta da subito la coverage massima ([figura 18](#), [figura 21](#)) e viene illustrata la mutation coverage ([figura 22](#)).

La mutation coverage dell'intera classe `WriteCache` è del 32% ([figura 12](#)).

## 3. Storm

### 3.1. Presentazione

Apache Storm è un sistema di calcolo distribuito in tempo reale, progettato per l'elaborazione efficiente di grandi quantità di dati ad alta velocità. Questa caratteristica gli consente di gestire oltre un milione di record al secondo per nodo all'interno di cluster di dimensioni ridotte. Il sistema è noto per la sua scalabilità, tolleranza agli errori e affidabilità.

Le topologie in Storm sono composte da due tipi di nodi: "Bolt" che elaborano gli stream di dati in ingresso e producono stream di dati in uscita, oltre a eseguire funzioni di filtraggio, aggregazione, unione di dati e comunicazione con database; "Spout" che agiscono come sorgenti di stream, generando sequenze di tuple, cioè elenchi ordinati di elementi che rappresentano i dati da elaborare all'interno del sistema.

### 3.2. Scelta delle classi e dei metodi

Le classi di test scelte per l'analisi sono *CoordinatedBolt* e *JoinBolt*, appartenenti rispettivamente ai package *coordination* e *bolt* all'interno del modulo *storm-client*.

La scelta di *CoordinatedBolt* è giustificata dalla sua presenza sin dalla prima versione del progetto e dal continuo flusso di commit nel corso del tempo. Questo fattore motiva la selezione della classe poiché le frequenti modifiche possono indicare la presenza potenziale di bug o problemi all'interno della classe.

Parallelamente, il test su *JoinBolt* è stato incluso per la caratteristica della classe di avere un valore di churn costantemente elevato nel corso del tempo, suggerendo l'aggiunta di nuove funzionalità in modo contiguo. L'analisi di *JoinBolt* potrebbe rivelare informazioni cruciali sulla gestione delle nuove funzionalità e sulla stabilità della classe nel lungo periodo.

L'approfondita esplorazione di *CoordinatedBolt* e *JoinBolt* permetterà una valutazione delle dinamiche di sviluppo, nonché della stabilità e dell'evoluzione del sistema *storm-client*.

### 3.3. CoordinatedBolt

Questa classe è una componente di tipo Bolt specializzata nel rilevare quando un altro Bolt ha ricevuto le tuple associate a un determinato request ID all'interno di una topologia di elaborazione dati in tempo reale.

In sintesi, *CoordinatedBolt* si occupa di: monitoraggio delle tuple, rilevamento del completion e coordinamento avanzato.

La classe di test è composta da *testPrepareAndExecute* e *testPrepareAndExecuteDiffConstructor*, nei quali vengono testati i metodi:

- `public CoordinatedBolt(IRichBolt delegate, String sourceComponent, SourceArgs sourceArgs, IdStreamSpec idStreamSpec)`: uno dei tre costruttori del SUT;
- `public CoordinatedBolt(IRichBolt delegate)`: uno dei tre costruttori del SUT;
- `public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector)`: serve per configurare la topologia dei vari oggetti di Storm e dell'output di ricezione delle tuple. Nei test, l'oggetto `collector` è istanziato tramite l'uso di Mock, in quanto non può assumere valore null o solleverebbe l'eccezione `NullPointerException`, e l'output del Bolt viene gestito da `MockBolt`;
- `public void execute(Tuple tuple)`: prende in input una tupla, la elabora e la emette nella topologia creata in fase di `prepare`;
- `private TupleType getTupleType(Tuple tuple)`: è un'enumerazione contenuta nel SUT;
- `private boolean checkFinishId(Tuple tup, TupleType type)`: verifica se un id specifico ha soddisfatto le condizioni di completamento nel contesto della coordinazione dei bolt.

*testPrepareAndExecute* verifica il corretto inserimento dei parametri `srcComponent` e `streamId` nella classe di test.

*testPrepareAndExecuteDiffConstructor* istanzia un Bolt delegato fittizio (`MockRichBoltTimeout`), il quale riceve tuple inviate dalla classe di test.

#### 3.3.1. Individuazione dominio di input e partizionamento dominio

- `public CoordinatedBolt(...)`: il primo costruttore della classe non è oggetto di test specifici, tuttavia viene testato implicitamente quando viene eseguito *testPrepareAndExecute*.

Analizzando i parametri di input, `delegate` viene istanziato come un oggetto `MockRichBoltTimeout` con lo scopo di stabilire un ambiente di test controllato, e questo prende un oggetto Bolt e deve avere un'istanza valida, senza poter assumere valore null, altrimenti solleverebbe l'eccezione di tipo

*NullPointerException*; *sourceComponent* serve ad associare un nome alla sorgente dati e può assumere valore valido o non valido, che verrà approfondito dopo; *sourceArgs* può essere anch'esso un'enumerazione di possibili argomenti associati alla sorgente di dati, il cui valore è specificato successivamente; *idStreamSpec* rappresenta le specifiche di identificazione dello stream e il valore assunto viene specificato in seguito.

Entrando nel particolare, i possibili valori assegnati inizialmente alle classi di equivalenza sono:

```
srcComponent = {"srcTest", ""}  
sourceArgs = {CoordinatedBolt.SourceArgs.all()}  
idStreamSpec = {CoordinatedBolt.IdStreamSpec.makeDetectSpec(..., ...)},
```

dove i parametri di *makeDetectSpec* sono stringhe che possono assumere qualsiasi valore, purché siano uguali a quelle inserite tramite *TopologyContext*, e il parametro si riferisce a una classe interna alla classe di test che ha il compito di creare un oggetto *GlobalStreamId*, il quale servirà poi a creare un riferimento allo stream di output del Bolt, prima di dichiarare la topologia.

Per ogni classe di equivalenza, il test mira a verificare che il valore del *GlobalStreamId* di *idStreamSpec* corrisponda ai valori di *this.srcComponent* e *this.StreamId*.

In generale, dal momento che le operazioni di *prepare* ed *execute* potrebbero avere effetti collaterali o influire su altri aspetti del SUT, il test verifica anche il corretto comportamento di queste due operazioni oltre al confronto specifico di *idStreamSpec*.

Analizzando le possibili combinazioni tra i parametri, vengono prodotti i casi di test.

[Casi di test] - Per quanto riguarda i risultati attesi per ognuno dei cinque casi: nei primi due casi ci si aspetta che la chiamata a *prepare* imposti correttamente l'*idStreamSpec* con il risultato di *makeDetectSpec()*; nel terzo caso il risultato atteso è che la chiamata a *prepare* imposti l'*idStreamSpec* con uno *streamId* vuoto; e anche negli ultimi due casi il valore di *streamId* non dovrebbe dare problemi né sollevare eccezioni durante l'inizializzazione dell'oggetto Bolt.

Analizzando la coverage, il risultato di questi test sul costruttore porta da subito a una statement coverage e una branch coverage del 100% ([figura 23](#)).

Quindi, il test sul costruttore è già perfettamente accettabile.

In secondo luogo, per provare ad aumentare la coverage degli altri metodi della classe, le partizioni del dominio vengono estese, e le nuove possibili classi di equivalenza sono:

```
srcComponent = {"srcTest", "", null}  
sourceArgs = {CoordinatedBolt.SourceArgs.all(), CoordinatedBolt.SourceArgs.single()}  
idStreamSpec = {CoordinatedBolt.IdStreamSpec.makeDetectSpec(..., ...), null}
```

A questo punto, analizzando le possibili combinazioni tra i parametri, vengono aggiunti cinque casi di test ai precedenti.

[Casi di test] - Ora nei primi quattro casi ci si aspetta che la chiamata a *prepare* imposti correttamente *sourceArgs* su *single*; mentre dove *idStreamSpec* è *null* ci si aspetta che i parametri nulli vengano gestiti dalla classe *CoordinatedBolt* senza generare eccezioni durante la creazione dell'istanza.

L'ultimo caso è stato aggiunto per avere almeno una situazione in cui siano combinati i valori null di *idStreamSpec* e il valore *all()* di *SourceArgs*, così da non avere delle configurazioni completamente unidimensionali per quanto riguarda questi due parametri.

Come già detto, l'aggiunta di questi nuovi valori ai parametri di test non avrà effetto sulla coverage dei test sul costruttore, che aveva già raggiunto il valore massimo, ma serviranno per i metodi successivi.

- *public CoordinatedBolt(...)*: anche il secondo costruttore è testato implicitamente, questa volta durante *testPrepareAndExecuteDiffConstructor*.

In questo caso il metodo ha un solo parametro, cioè *delegate*, e la fase di analisi per la scelta dei parametri è inutile perché viene usato solo il parametro istanziato come oggetto *MockRichBoltTimeout*. Data la semplicità di questo metodo, il test implicito basta per garantire una coverage del 100% ([figura 23](#)).

- *public void prepare(...)*: effettuando uno studio dei parametri, il primo rappresenta la configurazione passata al bolt ed è un oggetto *HashMap<>* vuoto, in quanto ai fini del test non c'era bisogno di aggiungere particolari parametri di configurazione; il secondo parametro fornisce informazioni sul contesto della topologia, e viene istanziato tramite l'utilizzo del framework *Mockito*, con cui vengono definiti dei comportamenti volti ad aumentare la coverage dei metodi contenuti nel SUT; l'ultimo parametro è responsabile della raccolta dei risultati delle tuple e l'oggetto *collector* è istanziato

anch'esso tramite l'uso di Mock, in quanto non può assumere valore null o solleverebbe un'eccezione di tipo *NullPointerException*, e l'output del Bolt viene gestito da *MockBolt*.

I parametri *config* e *collector* vengono creati sempre allo stesso modo, ma analizzando il parametro *context*, si nota che questo è oggetto di alcune operazioni prima della chiamata a *prepare*.

Nella funzione, i parametri che interessano le operazioni su *context* sono *srcComponent*, *streamId* e *topologyTimeout*. Le partizioni iniziali per questi parametri sono:

```
srcComponent = {stringa vuota, stringa non vuota}
streamId = {stringa vuota, stringa non vuota}
topologyTimeout = {>=0, < 0, null},
```

dove il valore null non è specificato nei casi in *@Parameterized.Parameters* ma viene utilizzato direttamente in *testPrepareAndExecuteDiffConstructor*.

Da queste partizioni, le classi di equivalenza generate per i tre parametri sono:

```
srcComponent = {"srcTest", ""}
streamId = {"streamTest", "", Constants.COORDINATED_STREAM_ID}
topologyTimeout = {-1, 0, 10}
```

La scelta dei valori per *topologyTimeout* è stata guidata da un'analisi boundary values per la scelta dei valori -1 e 0, a cui è stato aggiunto un valore più lontano dallo zero.

**[Casi di test]** - L'output atteso per l'operazione *prepare* dovrebbe essere un successo per tutti i casi di test, poiché non ci sono mai configurazioni problematiche o problemi che potrebbero causare errori perché mal gestiti.

Quando *topologyTimeout* è -1 il risultato è che *numSourceReports* sarà nullo, quando il parametro è 0 o 10, *numSourceReports* assume il valore di *topologyTimeout*.

In tutti i casi non ci sono problemi nella configurazione.

Nella situazione iniziale, la statement coverage è al 73% e la branch coverage al 66% ([figura 23](#), [figura 24](#)), mentre la data flow coverage è al 70% ([figura 25](#)).

Inoltre viene riportata la mutation coverage, dal report di Pit, per *prepare* in questa prima situazione ([figura 26](#)).

Riprendendo la descrizione del costruttore, in un secondo momento sono state aggiunte nuove combinazioni di parametri, che in questo caso impatteranno la bontà del test.

Per quanto riguarda i parametri usati in *prepare*, ciò che ci interessa sono le nuove partizioni di *srcComponent*, *streamId*, *topologyTimeout* e anche *sourceArgs* che viene chiamato all'interno di *prepare*.

In particolare, i nuovi casi introdotti testeranno il metodo anche per valori null di *srcComponent* e *topologyTimeout* (che quindi potrà assumere il valore null anche in *testPrepareAndExecute*), e il valore *single()* di *sourceArgs*.

**[Casi di test]** - Il caso particolare tra questi è quello in cui *topologyTimeout* è null, per cui la topologia potrebbe continuare a eseguirsi senza limite di tempo.

Tuttavia, questa situazione viene gestita inizializzando il valore della mappa *TimeCacheMap* con il valore di timeout predefinito.

Dopo l'aggiunta dei nuovi valori, la statement coverage e la branch coverage sono aumentate rispettivamente all'81% e al 75% ([figura 27](#), [figura 28](#)).

Per quanto riguarda la branch coverage, questa raggiunge il 76% ([figura 29](#)).

Venne riportata la mutation coverage del metodo dopo la modifica ([figura 30](#)).

- *public void execute(...)*: l'unico parametro di input *tuple* viene istanziato usando la classe *TupleImpl* di Storm e il suo costruttore *public TupleImpl(GeneralTopologyContext context, List<Object> values, String srcComponent, int taskId, String streamId)*.

All'interno del costruttore, l'oggetto *context* viene istanziato con campi *fields* che identificano i dati nella tupla; la lista *values* contiene i dati effettivi da inserire nella tupla; *srcComponent* e *streamId* devono essere identici sia per il costruttore del SUT sia per il *context*; *taskId* può assumere qualsiasi valore intero maggiore o uguale a zero.

Per i parametri che vengono fatti variare nei diversi casi di test, le partizioni iniziali sono:

```
srcComponent = {stringa vuota, stringa non vuota}
streamId = {stringa vuota, stringa non vuota}
values (data) = {objectList1}
```

In particolare quest'ultimo parametro viene creato in questo modo:

```
List<Object> objectList1 = new ArrayList<>();  
objectList1.add("");  
objectList1.add(30);
```

Da queste partizioni, le classi di equivalenza generate per i tre parametri sono:

```
srcComponent = {"srcTest", ""}  
streamId = {"streamTest", "", Constants.COORDINATED_STREAM_ID}  
values (data) = {objectList1}
```

[Casi di test] - La diversa configurazione dei valori dei parametri determina il tipo di tupla (in base all'implementazione di `getTupleType`), in base a cui vengono eseguite operazioni di tracciamento e poi viene chiamato `checkFinishId`.

Se il tipo è ID, il metodo aggiorna lo stato interno del tracking per indicare che l'id è stato ricevuto; se è COORD, viene indicata una segnalazione ricevuta e si aggiorna il conteggio previsto delle tuple; se il tipo è REGULAR, la tupla viene passata al metodo `execute` del bolt delegato.

Nella situazione iniziale la statement coverage è al 70% e la branch coverage al 75% ([figura 23](#), [figura 31](#)). La data flow coverage è al 61.5% ([figura 32](#)). Viene riportata la mutation coverage del metodo ([figura 33](#)).

Anche in questo caso vengono introdotti dei miglioramenti sui test aggiungendo nuovi parametri e nuove combinazioni, provando ad aumentare la coverage per il metodo `execute`.

Innanzitutto i nuovi casi testano il caso in cui `srcComponent` è null e testano una nuova lista in `values` che viene creata in questo modo:

```
List<Object> objectList2 = new ArrayList<>();  
objectList2.add("test");  
objectList2.add(20);
```

A seguito di questa modifica, si generano altre cinque combinazioni di parametri.

[Casi di test] - A differenza del metodo `prepare`, dopo l'aggiunta di questi casi la coverage per il metodo `execute` rimane invariata.

Allora vengono aggiunti ulteriori cinque casi di test, che accentuano la multidimensionalità, con lo scopo di migliorare la coverage di tutti i tipi.

Il risultato sperato viene pienamente raggiunto dopo questa modifica.

[Casi di test] - la statement coverage di `execute` aumenta al 100%, la branch coverage all'87% ([figura 34](#), [figura 35](#)) e la data flow coverage all'88% ([figura 36](#)). Viene riportata anche la mutation coverage dopo la modifica ai test ([figura 37](#)).

Visti in questo modo, alcuni degli ultimi casi sembrano uguali ad altri casi precedenti, ma è importante sottolineare che all'interno del metodo `execute` in `CoordinatedBolt` viene utilizzato anche il valore di `idStreamSpec`. Inoltre sempre in `execute` viene chiamato il metodo `checkFinishId`, che a sua volta sfrutta il valore del parametro `sourceArgs`.

Nei casi aggiunti alla fine, hanno una notevole importanza le variazioni a questi ultimi due parametri. Infatti, come accennato in precedenza, ciò che aiuta a migliorare la coverage è l'aumento delle combinazioni tra parametri, azione che porta a un approccio multidimensionale, ma senza arrivare alla cardinalità di un prodotto cartesiano.

I valori di `idStreamSpec` e `sourceArgs` non sono stati inseriti nella lista delle classi di equivalenza perché non sono propriamente parametri di input del metodo `execute`, ma appunto ne viene marcata la rilevanza.

- `private TupleType getTupleType(...)` & `private boolean checkFinishId(...)`: questi ultimi due metodi vengono entrambi testati implicitamente all'interno del metodo `execute`. Entrambi prendono come input lo stesso input di `execute`, cioè una tupla, e `checkFinishId` ha come parametro anche il risultato di `getTupleType`.

Per questi due metodi non verrà fatto uno studio esaustivo come per i precedenti, anche perché i parametri seguono lo studio già svolto per `execute`.

E però interessante sapere come varia la coverage anche per loro nei tre momenti che si susseguono, aggiungendo ogni volta cinque nuovi casi di test.

Per semplicità queste informazioni vengono riportate in una [tabella](#).

Si nota che gli ultimi cinque casi sono gli unici che portano miglioramenti da entrambe le parti.

Nelle figure 38 e 39 sono illustrate le differenze di statement e branch coverage del metodo `getTupleType`, dalle figure 40, 41 e 42 si nota l'aumento della data flow coverage, che passa dal 52% all'82%. Per `getTupleType` è riportata la mutation coverage (figura 44).

Infine, si nota che la mutation coverage passa dal 13% (figura 45) al 15% (figura 46) grazie alla prima modifica, e poi rimane la stessa per l'ultima aggiunta di parametri.

Per riassumere tutti i parametri, che non vengono mai citati insieme, questo [screen](#) li mostra tutti, nell'ordine in cui vengono usati durante i test.

### 3.4. JoinBolt

Questa classe viene utilizzata per eseguire operazioni di join su tuple provenienti da più stream all'interno di una topologia di Storm.

La classe ha due costruttori principali, metodi che specificano le clausole di join, metodi per la gestione delle finestre temporali e altri metodi di utilità per eseguire join interni e left join, per la proiezione dei campi, e per ottenere il valore di un campo all'interno di una tupla.

In generale la classe fornisce un modo flessibile per eseguire operazione di join su dati distribuiti all'interno di una topologia di Storm.

Il test viene effettuato all'interno della classe `TestJoinBolt` e nelle due classi `TestJoinBoltJoin` e `TestJoinBoltSelect`, che testano rispettivamente le operazioni di join e select sui bolt.

In particolare, i metodi testati sono:

- `public JoinBolt(Selector type, String srcOrStreamId, String fieldName)`: è il costruttore;
- `public JoinBolt leftJoin(String newStream, String field, String priorStream)`: esegue un left join con un nuovo stream, basandosi su field specifici;
- `public JoinBolt join(String newStream, String field, String priorStream)`: esegue un inner join con un nuovo stream;
- `public JoinBolt select(String commaSeparatedKeys)`: consente di specificare i campi da includere nell'output dei bolt.

#### 3.4.1. Individuazione dominio di input e partizionamento dominio

- `public JoinBolt(...)`: il dominio di input è stato partizionato inizialmente in questo modo:

```
type = {null, valid instance}
srcOrStreamId = {null, stringa vuota, stringa non vuota}
fieldName = {null, stringa vuota, stringa non vuota}
```

I possibili valori per le classi di equivalenza sono:

```
type = {null, Selector.STREAM, Selector.SOURCE}
srcOrStreamId = {null, "", 'streamName'}
fieldName = {null, "", 'keyField'}
```

Per ogni classe di equivalenza, il test verifica se la creazione di un oggetto `JoinBolt` con i parametri forniti produce il risultato atteso, passato anch'esso come parametro.

Quest'ultimo assume valori come `JoinBolt.class` e `NullPointerException.class`.

Considerando anche il parametro `expectedResult`, sono riportate le combinazioni di parametri utilizzate per sviluppare i casi di test.

[[Casi di test](#)] - Nei casi 1, 2, 4 e 5 ci si aspetta che il risultato sia un'istanza della classe `JoinBolt` (l'ultimo parametro indica l'expected result). Nel primo caso tutti i parametri del costruttore sono scelti in modo da essere validi (le stringhe non sono nulle); nel secondo caso, nonostante il selettore è null, mi aspetto che il costruttore gestisca tutto correttamente, e il risultato è un'istanza di `JoinBolt` perché le due stringhe sono valide; nel quarto e nel quinto caso non ho nessun parametro nullo.

L'unico caso in cui viene sollevata un'eccezione è il terzo, perché entrambi i parametri sono null.

La coverage del costruttore raggiunge il 100% con i casi di test scelti ([figura 47](#)), che quindi coprono esaustivamente il metodo.

- `public JoinBolt leftJoin(...)`: il dominio di input viene partizionato inizialmente in questo modo:

```
newStream = {stream nullo, stream non nullo}
field = {null, stringa valida = campo in newStream, stringa valida != campo in newStream}
priorStream = {null, stringa valida = stream esistente, stringa valida = stream non esistente}
```

Il valore di questi parametri dipende dai parametri creati in [@Parameterized.Parameters](#). In particolare, il valore di *newStream* dipende dalla combinazione di *secondStream* e *field2Index*. Di conseguenza, in una prima analisi, i possibili valori scelti per le classi di equivalenza derivano da:

*newStream* = *StreamGenerator.createStream(secondStream, field2Index).streamName*, dove il valore di questi due parametri appartiene alle seguenti classi di equivalenza:

```
secondStream = {STREAM.RESERVATIONS, STREAM.ORDERS, STREAM.NULL,
                STREAM.EMPTY}
field2Index = {0, 1}
```

```
field = {"foo", stream2.streamFields[stream2.fieldIndex]}
priorStream = {StreamGenerator.createStream(STREAM.MENU, 0).streamName}
```

Il *leftJoin* viene usato nei casi in cui il *jointype* è LEFT o WRONG\_LEFT. In entrambi i casi, dopo una serie di funzioni applicate sul bolt, viene confrontato l'*expected result* con un parametro chiamato *actualSize*, inizializzato tramite la classe *CustomCollector*, che ha un comportamento personalizzato rispetto all'emissione di tuple, tenendo traccia delle tuple emesse dal bolt e memorizzandole in una *ArrayList* di cui ci interessa la dimensione.

Il valore del parametro *expectedResult* viene sempre inserito nella lista di parametri creati in [@Parameterized.Parameters](#) e varia in base al resto degli input.

Per questa prima scelta dei valori dei parametri viene fatta un'analisi della coverage, e da questa risulta che, per il metodo *doLeftJoin*, la statement coverage è del 67%, la branch coverage del 40% ([figura 47](#)) e la data flow coverage del 52% ([figura 49](#)).

In secondo luogo, analizzando meglio le funzionalità di *TestJoinBolt* e in particolare dei metodi di join, è stato aggiunto un terzo stream i cui valori vengono sempre presi da [@Parameterized.Parameters](#), utilizzando questa volta una lista di parametri diversa, in cui il valore di *thirdStream* e *field3Index* sono diversi da null e 0.

Una conseguenza di questa aggiunta è un nuovo valore per i parametri *newStream* e *field*:

```
newStream = {stream2.streamName, stream3.streamName}
field = {"foo", stream2.streamFields[stream2.fieldIndex], stream3.streamFields[stream3.fieldIndex]}
```

In questo secondo caso, il test funziona allo stesso modo, e viene solo concatenato un *leftJoin* con *stream3* al primo *leftJoin* nel caso in cui *jointype* è LEFT.

Analizzando la coverage, si vede che rimane invariata.

Inoltre viene riportata anche la mutation coverage dal report di Pit per *doLeftJoin* ([figura 51](#)).

- *public JoinBolt join(...)*: il dominio di input viene partizionato come per il metodo precedente:

```
newStream = {stream nullo, stream non nullo}
field = {null, stringa valida = campo in newStream, stringa valida != campo in newStream}
priorStream = {null, stringa valida = stream esistente, stringa valida = stream non esistente}
```

Anche in questo caso, il valore di questi parametri dipende dai parametri creati in [@Parameterized.Parameters](#). In particolare, il valore di *newStream* dipende dalla combinazione di *secondStream* e *field2Index*.

Adesso l'analisi per la scelta delle classi di equivalenza è leggermente diversa dal caso precedente, e i valori scelti derivano da:

*newStream* = *{StreamGenerator.createStream(secondStream, field2Index).streamName}*, null, “ ”, dove il valore dei due parametri di *createStream* appartiene alle seguenti classi di equivalenza:

```
secondStream = {STREAM.RESERVATIONS, STREAM.ORDERS, STREAM.NULL,
                STREAM.EMPTY}
field2Index = {0, 1}
```

```
field = {" ", stream2.streamFields[stream2.fieldIndex]}
priorStream = {StreamGenerator.createStream(STREAM.MENU, 0).streamName, "foo"}
```

Il valore “ ” per *newStream* e *field* eseguono il test nel caso *EMPTY\_STRING\_JOIN*, mentre il valore “*foo*” per *priorStream* esegue il test nel caso *NOT\_EXISTING\_PRIOR*.

Quando *newStream* è null, ci si trova nel caso *EMPTY*.

Il *join* viene usato in questi casi sopra citati, e come per la funzione precedente, il risultato di alcune funzioni applicate al bolt viene confrontato con un expected result contenuto nella lista di parametri. Come per *leftJoin*, la prima scelta dei parametri non prevede *stream3*, che viene aggiunto in un secondo momento per valutare la differenza nei test.

Nel primo caso, concentrandosi sulla statement coverage e la branch coverage del metodo *doInnnerJoin*, queste sono rispettivamente al 66% e al 50% ([figura 47](#)), mentre la data flow coverage è del 94,7% ([figura 52](#)).

È stato evidenziato lo studio su questo metodo perché è l'unico che subisce un cambiamento con l'aggiunta di *stream3*.

Dopo aver aggiunto il terzo stream che prende i valori da [@Parameterized.Parameters](#), anche in questo caso si aggiunge una classe di equivalenza per i parametri *newStream* e *field* che ora risultano essere:

```
newStream = {stream2.streamName, stream3.streamName}  
field = {"", stream2.streamFields[stream2.fieldIndex], stream3.streamFields[stream3.fieldIndex]}
```

In questo secondo caso, il test funziona allo stesso modo, e viene solo concatenato un *join* con *stream3* al primo *join* nel caso in cui *jointype* è INNER.

Inoltre, l'analisi della coverage evidenzia che adesso per il metodo *doInnnerJoin* la statement coverage è del 100% e la branch coverage è dell'87% ([figura 53](#)), mentre la data flow coverage è rimasta invariata.

Viene riportata la mutation coverage dei metodi *doInnnerJoin* e *doJoin* ([figura 54](#), [figura 55](#)).

- *public JoinBolt select(...)*: i possibili valori del parametro appartengono alle seguenti partizioni scelte inizialmente:

```
commaSeparatedKeys = {istanza nulla, istanza non nulla}
```

Successivamente, le classi di equivalenza scelte sono state molteplici, come:

```
commaSeparatedKeys = {null, "field1,field2", "field1", "", "notExistingField!", "field1,field2,field3"}
```

[[Casi di test](#)] - Nel primo caso è rappresentato l'utilizzo di stringhe non vuote come chiavi di selezione e ci si aspetta che il metodo *select* le gestisca correttamente, producendo l'output atteso trasformando i dati dello stream; nel secondo caso c'è una singola chiave di selezione valida; nel terzo caso da una stringa vuota ci si aspetta l'output atteso di uno stream nullo; il quarto caso rappresenta il passaggio di valore null come chiave di selezione, che solleverà un'eccezione di tipo *NullPointerException*; nel quinto caso il valore non esistente assegnato a *commaSeparatedKeys* fa in modo che il metodo *select* gestisca questa situazione come si comporta quando riceve una stringa vuota; nell'ultimo caso l'expected result è la risultante della trasformazione dei dati dello stream, compresi campi nulli.

Il metodo *select* raggiunge la copertura massima con i casi di test scelti per quanto riguarda statement e branch coverage ([figura 47](#)), mentre arriva al 93,75% di data flow coverage ([figura 56](#)).

È riportata anche la mutation coverage per questo metodo ([figura 57](#)).

Un'importante conclusione che si può trarre è che, soprattutto a causa delle modifiche nei test per i metodi di *join*, il miglioramento maggiormente osservabile sta nel valore della mutation coverage totale della classe, che è aumentata dal 54% al 58% ([figura 58](#), [figura 59](#)).

Oltre a questo, come scritto durante l'analisi dei singoli metodi, l'aggiunta di *stream3* non ha provocato enormi differenze nei test.

Durante il test per la classe *JoinBolt* vengono testati anche i metodi *prepare* ed *execute*, per cui non viene riportato lo studio esplicito, ma il loro ruolo si deduce dagli altri metodi testati, soprattutto per *execute*, che al suo interno contiene tutte le funzioni riportate nello studio (effettua una chiamata a *hashJoin*, che chiama a sua volta *doJoin*, in cui compaiono i metodi analizzati a fondo *doLeftJoin* e *doInnnerJoin*).

## 4. Collegamenti

- GitHub: <https://github.com/tizianotaglienti/bookkeeper> , <https://github.com/tizianotaglienti/storm>
- Travis CI: <https://app.travis-ci.com/github/tizianotaglienti/bookkeeper> , <https://app.travis-ci.com/github/tizianotaglienti/storm>
- SonarCloud: [https://sonarcloud.io/project/overview?id=tizianotaglienti\\_bookkeeper4](https://sonarcloud.io/project/overview?id=tizianotaglienti_bookkeeper4) ,  
[https://sonarcloud.io/project/overview?id=tizianotaglienti\\_storm](https://sonarcloud.io/project/overview?id=tizianotaglienti_storm)
- Per il setup dell'ambiente si sono seguite le istruzioni al seguente link:  
<https://cwiki.apache.org/confluence/display/BOOKKEEPER/Developer+Setup>
- In locale, dopo aver avviato SonarQube da *StartSonar.bat*, i comandi utilizzati sono stati:  
*mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent verify*, per eseguire la build e lanciare i test ottenendo la coverage;  
*mvn clean verify -P badua-coverage-offline*, per eseguire Ba-Dua e ottenere i relativi report;  
*mvn clean verify -P pit-test*, per eseguire Pit e ottenere i relativi report.

## 5. Coverage (figure)

### ByteBufList

Element	Missed Instructions	Cov.	Missed Branches	Cov.
coalesce(ByteBufList)		0%		0%
clone(ByteBufList)		0%		0%
touch(Object)		0%		0%
hasArray()		0%		0%
array()		0%		n/a
arrayOffset()		0%		n/a
getBuffer(int)		0%		n/a
retain()		0%		n/a
size()		0%		n/a
getBytes(byte[])		100%		100%
deallocate()		100%		100%
readableBytes()		100%		100%
static {...}		100%		n/a
ByteBufList(Recycler.Handle)		100%		n/a
get(ByteBuf, ByteBuf)		100%		n/a
toArray()		100%		n/a
get()		100%		n/a
get(ByteBuf)		100%		n/a
add(ByteBuf)		100%		n/a
prepend(ByteBuf)		100%		n/a
Total	114 of 278	58%	10 of 18	44%

Figura 1

### ByteBufList.Encoder

Element	Missed Instructions	Cov.	Missed Branches	Cov.
write(ChannelHandlerContext, Object, ChannelPromise)		74%		87%
ByteBufList.Encoder(boolean)		100%		n/a
Total	17 of 72	76%	1 of 8	87%

Figura 2

```

@Override
public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception {
    if (msg instanceof ByteBufList) {
        ByteBufList b = (ByteBufList) msg;

        try {
            if (prependSize) {
                // Prepend the frame size before writing the buffer list, so that we only have 1 single size
                // header
                ByteBuf sizeBuffer = ctx.alloc().directBuffer(4, 4);
                sizeBuffer.writeInt(b.readableBytes());
                ctx.write(sizeBuffer, ctx.voidPromise());
            }

            // Write each buffer individually on the socket. The retain() here is needed to preserve the fact
            // that ByteBuf are automatically released after a write. If the ByteBufPair ref count is increased
            // and it gets written multiple times, the individual buffers refcount should be reflected as well.
            int buffersCount = b.buffers.size();
            for (int i = 0; i < buffersCount; i++) {
                ByteBuf bx = b.buffers.get(i);
                // Last buffer will carry on the final promise to notify when everything was written on the
                // socket
                ctx.write(bx.retainDuplicate(), i == (buffersCount - 1) ? promise : ctx.voidPromise());
            }
        } finally {
            ReferenceCountUtil.safeRelease(b);
        }
    } else {
        ctx.write(msg, promise);
    }
}

```

Figura 3

```

<class name="org/apache/bookkeeper/util/ByteBufList$Encoder">
<method name="write" desc="(Lio/netty/channel/ChannelHandlerContext;Ljava/
<du var="this" def="302" use="306" target="309" covered="0"/>
<du var="this" def="302" use="306" target="317" covered="1"/>
<du var="ctx" def="302" use="328" covered="1"/>
<du var="ctx" def="302" use="322" covered="1"/>
<du var="ctx" def="302" use="309" covered="0"/>
<du var="ctx" def="302" use="311" covered="0"/>
<du var="msg" def="302" use="302" target="303" covered="1"/>
<du var="msg" def="302" use="302" target="328" covered="1"/>
<du var="msg" def="302" use="328" covered="1"/>
<du var="msg" def="302" use="303" covered="1"/>
<du var="promise" def="302" use="328" covered="1"/>
<du var="this.prependSize" def="302" use="306" target="309" covered="0"/>
<du var="this.prependSize" def="302" use="306" target="317" covered="1"/>
<du var="b" def="303" use="317" covered="1"/>
<du var="b" def="303" use="325" covered="1"/>
<du var="b" def="303" use="319" covered="1"/>
<du var="b" def="303" use="310" covered="0"/>
<du var="buffersCount" def="317" use="318" target="319" covered="1"/>
<du var="buffersCount" def="317" use="318" target="325" covered="1"/>
<du var="buffersCount" def="317" use="322" target="322" covered="1"/>
<du var="buffersCount" def="317" use="322" target="322" covered="1"/>
<du var="i" def="318" use="318" target="319" covered="1"/>
<du var="i" def="318" use="318" target="325" covered="0"/>
<du var="i" def="318" use="319" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="318" covered="1"/>
<du var="bx" def="319" use="322" covered="1"/>
<du var="i" def="318" use="318" target="319" covered="1"/>
<du var="i" def="318" use="318" target="325" covered="1"/>
<du var="i" def="318" use="319" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="0"/>
<du var="i" def="318" use="318" covered="1"/>
<counter type="DU" missed="7" covered="27"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 4

## ByteBufList.Encoder

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• <a href="#">write(ChannelHandlerContext, Object, ChannelPromise)</a>		100%		100%
• <a href="#">ByteBufList.Encoder(boolean)</a>		100%		n/a
Total	0 of 72	100%	0 of 8	100%

Figura 5

```

<class name="org/apache/bookkeeper/util/ByteBufList$Encoder">
<method name="write" desc="(Lio/netty/channel/ChannelHandlerContext;Ljava/
<du var="this" def="302" use="306" target="309" covered="1"/>
<du var="this" def="302" use="306" target="317" covered="1"/>
<du var="ctx" def="302" use="328" covered="1"/>
<du var="ctx" def="302" use="322" covered="1"/>
<du var="ctx" def="302" use="309" covered="1"/>
<du var="ctx" def="302" use="311" covered="1"/>
<du var="msg" def="302" use="302" target="303" covered="1"/>
<du var="msg" def="302" use="302" target="328" covered="1"/>
<du var="msg" def="302" use="328" covered="1"/>
<du var="msg" def="302" use="303" covered="1"/>
<du var="promise" def="302" use="328" covered="1"/>
<du var="this.prependSize" def="302" use="306" target="309" covered="1"/>
<du var="this.prependSize" def="302" use="306" target="317" covered="1"/>
<du var="b" def="303" use="317" covered="1"/>
<du var="b" def="303" use="325" covered="1"/>
<du var="b" def="303" use="319" covered="1"/>
<du var="b" def="303" use="310" covered="1"/>
<du var="buffersCount" def="317" use="318" target="319" covered="1"/>
<du var="buffersCount" def="317" use="318" target="325" covered="1"/>
<du var="buffersCount" def="317" use="322" target="322" covered="1"/>
<du var="buffersCount" def="317" use="322" target="322" covered="1"/>
<du var="i" def="318" use="318" target="319" covered="1"/>
<du var="i" def="318" use="318" target="325" covered="0"/>
<du var="i" def="318" use="319" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="318" covered="1"/>
<du var="bx" def="319" use="322" covered="1"/>
<du var="i" def="318" use="318" target="319" covered="1"/>
<du var="i" def="318" use="318" target="325" covered="1"/>
<du var="i" def="318" use="319" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="1"/>
<du var="i" def="318" use="322" target="322" covered="0"/>
<du var="i" def="318" use="318" covered="1"/>
<counter type="DU" missed="2" covered="32"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

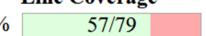
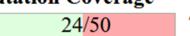
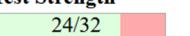
```

Figura 6

## Pit Test Coverage Report

### Package Summary

**org.apache.bookkeeper.util**

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	72% 	48% 	75% 

### Breakdown by Class

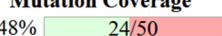
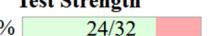
Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">ByteBufList.java</a>	72% 	48% 	75% 

Figura 7

# Pit Test Coverage Report

## Package Summary

**org.apache.bookkeeper.util**

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	76% 60/79	46% 23/50	72% 23/32

## Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">ByteBufList.java</a>	76% 60/79	46% 23/50	72% 23/32

Figura 8

## ReadCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
<a href="#">put(long, long, ByteBuf)</a>	42%	58%	25%	75%	2	3	11	21	0	1
<a href="#">size()</a>	0%	100%	0%	100%	4	4	9	9	1	1
<a href="#">count()</a>	0%	100%	0%	100%	2	2	6	6	1	1
<a href="#">ReadCache(ByteBufAllocator, long, int)</a>	100%	100%	100%	100%	0	2	0	12	0	1
<a href="#">get(long, long)</a>	100%	100%	100%	100%	0	3	0	13	0	1
<a href="#">ReadCache(ByteBufAllocator, long)</a>	100%	100%	n/a	n/a	0	1	0	2	0	1
<a href="#">close()</a>	100%	100%	n/a	n/a	0	1	0	2	0	1
Total	152 of 356	57%	11 of 18	38%	8	16	26	65	2	7

Figura 9

```

public ByteBuf get(long ledgerId, long entryId) {
    lock.readLock().lock();

    try {
        // We need to check all the segments, starting from the current one and looking
        // backward to minimize the
        // checks for recently inserted entries
        int size = cacheSegments.size();
        for (int i = 0; i < size; i++) {
            int segmentIdx = (currentSegmentIdx + (size - i)) % size;

            LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
            if (res != null) {
                int entryOffset = (int) res.first;
                int entryLen = (int) res.second;

                ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
                entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
                return entry;
            }
        }
    } finally {
        lock.readLock().unlock();
    }

    // Entry not found in any segment
    return null;
}

```

Figura 10

```

<class name="org/apache/bookkeeper/bookie/storage/ldb/ReadCache">
<method name="get" desc="(JJ)Lio/netty/buffer/ByteBuf;">
<du var="this" def="130" use="151" covered="1"/>
<du var="this" def="130" use="138" covered="1"/>
<du var="this" def="130" use="140" covered="1"/>
<du var="this" def="130" use="145" covered="1"/>
<du var="this" def="130" use="146" covered="1"/>
<du var="this" def="130" use="151" covered="1"/>
<du var="ledgerId" def="130" use="140" covered="1"/>
<du var="entryId" def="130" use="140" covered="1"/>
<du var="this.lock" def="130" use="151" covered="1"/>
<du var="this.lock" def="130" use="151" covered="1"/>
<du var="this.cacheSegments" def="130" use="146" covered="1"/>
<du var="this.currentSegmentIdx" def="130" use="138" covered="1"/>
<du var="this.cacheIndexes" def="130" use="140" covered="1"/>
<du var="thisallocator" def="130" use="145" covered="1"/>
<du var="size" def="136" use="137" target="138" covered="1"/>
<du var="size" def="136" use="137" target="151" covered="1"/>
<du var="size" def="136" use="138" covered="1"/>
<du var="i" def="137" use="137" target="138" covered="1"/>
<du var="i" def="137" use="137" target="151" covered="0"/>
<du var="i" def="137" use="138" covered="1"/>
<du var="i" def="137" use="137" covered="1"/>
<du var="segmentIdx" def="138" use="146" covered="1"/>
<du var="res" def="140" use="141" target="142" covered="1"/>
<du var="res" def="140" use="141" target="137" covered="1"/>
<du var="res" def="140" use="142" covered="1"/>
<du var="res" def="140" use="143" covered="1"/>
<du var="res.first" def="140" use="142" covered="1"/>
<du var="res.second" def="140" use="143" covered="1"/>
<du var="i" def="137" use="137" target="138" covered="1"/>
<du var="i" def="137" use="137" target="151" covered="1"/>
<du var="i" def="137" use="138" covered="1"/>
<du var="i" def="137" use="137" covered="1"/>
<counter type="DU" missed="1" covered="31"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 11

## Pit Test Coverage Report

### Package Summary

**org.apache.bookkeeper.bookie.storage.ldb**

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	58% <div style="width: 58%;">110/191</div>	27% <div style="width: 27%;">33/122</div>	49% <div style="width: 49%;">33/67</div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">ReadCache.java</a>	59% <div style="width: 59%;">41/70</div>	18% <div style="width: 18%;">8/44</div>	40% <div style="width: 40%;">8/20</div>
<a href="#">WriteCache.java</a>	57% <div style="width: 57%;">69/121</div>	32% <div style="width: 32%;">25/78</div>	53% <div style="width: 53%;">25/47</div>

Figura 12

```

public ByteBuf get(long ledgerId, long entryId) {
    1 lock.readLock().lock();

    try {
        // We need to check all the segments, starting from the current one and looking
        // backward to minimize the
        // checks for recently inserted entries
        int size = cacheSegments.size();
    2 for (int i = 0; i < size; i++) {
    3     int segmentIdx = (currentSegmentIdx + (size - i)) % size;

        LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
    1     if (res != null) {
            int entryOffset = (int) res.first;
            int entryLen = (int) res.second;

            ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
            entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
    1         return entry;
        }
    } finally {
    1     lock.readLock().unlock();
    }
}

// Entry not found in any segment
return null;
}

```

Figura 13

## WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• <a href="#">forEach(WriteCache.EntryConsumer)</a>		0%		0%
• <a href="#">lambda\$foreach\$0(long, long, long, long)</a>		0%		0%
• <a href="#">getLastEntry(long)</a>		0%		0%
• <a href="#">isEmpty()</a>		0%		0%
• <a href="#">deleteLedger(long)</a>		0%		n/a
• <a href="#">size()</a>		0%		n/a
• <a href="#">count()</a>		0%		n/a
• <a href="#">put(long, long, ByteBuf)</a>		96%		62%
• <a href="#">WriteCache(ByteBufAllocator, long, int)</a>		98%		66%
• <a href="#">get(long, long)</a>		100%		100%
• <a href="#">clear()</a>		100%		n/a
• <a href="#">close()</a>		100%		100%
• <a href="#">alignToPowerOfTwo(long)</a>		100%		n/a
• <a href="#">static {...}</a>		100%		n/a
• <a href="#">align64(int)</a>		100%		n/a
• <a href="#">WriteCache(ByteBufAllocator, long)</a>		100%		n/a
Total	265 of 617	57%	25 of 38	34%

Figura 14

```

public boolean put(long ledgerId, long entryId, ByteBuf entry) {
    int size = entry.readableBytes();

    // Align to 64 bytes so that different threads will not contend the same L1
    // cache line
    int alignedSize = align64(size);

    long offset;
    int localOffset;
    int segmentIdx;

    while (true) {
        offset = cacheOffset.getAndAdd(alignedSize);
        localOffset = (int) (offset & segmentOffsetMask);
        segmentIdx = (int) (offset >> segmentOffsetBits);

        if ((offset + size) > maxCacheSize) {
            // Cache is full
            return false;
        } else if (maxSegmentsize - localOffset < size) {
            // If an entry is at the end of a segment, we need to get a new offset and try
            // again in next segment
            continue;
        } else {
            // Found a good offset
            break;
        }
    }

    cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());

    // Update last entryId for ledger. This logic is to handle writes for the same
    // ledger coming out of order and from different thread, though in practice it
    // should not happen and the compareAndSet should be always uncontended.
    while (true) {
        long currentLastEntryId = lastEntryMap.get(ledgerId);
        if (currentLastEntryId > entryId) {
            // A newer entry is already there
            break;
        }

        if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
            break;
        }
    }

    index.put(ledgerId, entryId, offset, size);
    cacheCount.increment();
    cacheSize.addAndGet(size);
    return true;
}

```

Figura 15

```

<method name="put" desc="(J)Lio/netty/buffer/ByteBuf;)Z">
<du var="this" def="132" use="143" covered="1"/>
<du var="this" def="132" use="144" covered="1"/>
<du var="this" def="132" use="145" covered="1"/>
<du var="this" def="132" use="147" target="149" covered="1"/>
<du var="this" def="132" use="147" target="150" covered="1"/>
<du var="this" def="132" use="150" target="153" covered="0"/>
<du var="this" def="132" use="150" target="160" covered="1"/>
<du var="this" def="132" use="160" covered="1"/>
<du var="this" def="132" use="166" covered="1"/>
<du var="this" def="132" use="172" target="173" covered="1"/>
<du var="this" def="132" use="172" target="175" covered="0"/>
<du var="this" def="132" use="177" covered="1"/>
<du var="this" def="132" use="178" covered="1"/>
<du var="this" def="132" use="179" covered="1"/>
<du var="ledgerId" def="132" use="166" covered="1"/>
<du var="ledgerId" def="132" use="172" target="173" covered="1"/>
<du var="ledgerId" def="132" use="172" target="175" covered="0"/>
<du var="ledgerId" def="132" use="177" covered="1"/>
<du var="entryId" def="132" use="167" target="169" covered="0"/>
<du var="entryId" def="132" use="167" target="172" covered="1"/>
<du var="entryId" def="132" use="172" target="173" covered="1"/>
<du var="entryId" def="132" use="172" target="175" covered="0"/>
<du var="entryId" def="132" use="177" covered="1"/>
<du var="entry" def="132" use="160" covered="1"/>
<du var="this.cacheOffset" def="132" use="143" covered="1"/>
<du var="this.segmentOffsetMask" def="132" use="144" covered="1"/>
<du var="this.segmentOffsetBits" def="132" use="145" covered="1"/>
<du var="this.maxCacheSize" def="132" use="147" target="149" covered="1"/>
<du var="this.maxCacheSize" def="132" use="147" target="150" covered="1"/>
<du var="this.maxSegmentSize" def="132" use="150" target="153" covered="0"/>
<du var="this.maxSegmentSize" def="132" use="150" target="160" covered="1"/>
<du var="this.cacheSegments" def="132" use="160" covered="1"/>
<du var="this.lastEntryMap" def="132" use="166" covered="1"/>
<du var="this.lastEntryMap" def="132" use="172" target="173" covered="1"/>
<du var="this.lastEntryMap" def="132" use="172" target="175" covered="0"/>
<du var="this.index" def="132" use="177" covered="1"/>
<du var="this.cacheCount" def="132" use="178" covered="1"/>
<du var="this.cacheSize" def="132" use="179" covered="1"/>
<du var="size" def="132" use="147" target="149" covered="1"/>
<du var="size" def="132" use="147" target="150" covered="1"/>
<du var="size" def="132" use="150" target="153" covered="0"/>
<du var="size" def="132" use="150" target="160" covered="1"/>
<du var="size" def="132" use="177" covered="1"/>
<du var="size" def="132" use="179" covered="1"/>
<du var="alignedSize" def="136" use="143" covered="1"/>
<du var="offset" def="143" use="147" target="149" covered="1"/>
<du var="offset" def="143" use="147" target="150" covered="1"/>
<du var="offset" def="143" use="177" covered="1"/>
<du var="localOffset" def="144" use="150" target="153" covered="0"/>
<du var="localOffset" def="144" use="150" target="160" covered="1"/>
<du var="localOffset" def="144" use="160" covered="1"/>
<du var="segmentIdx" def="145" use="160" covered="1"/>
<du var="currentLastEntryId" def="166" use="167" target="169" covered="0"/>
<du var="currentLastEntryId" def="166" use="167" target="172" covered="1"/>
<du var="currentLastEntryId" def="166" use="172" target="173" covered="1"/>
<du var="currentLastEntryId" def="166" use="172" target="175" covered="0"/>
<counter type="DU" missed="11" covered="45"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 16

```

public boolean put(long ledgerId, long entryId, ByteBuf entry) {
    int size = entry.readableBytes();

    // Align to 64 bytes so that different threads will not contend the same L1
    // cache line
    int alignedSize = align64(size);

    long offset;
    int localOffset;
    int segmentIdx;

    while (true) {
        offset = cacheOffset.getAndAdd(alignedSize);
        localOffset = (int) (offset & segmentOffsetMask);
        segmentIdx = (int) (offset >>> segmentOffsetBits);

        if ((offset + size) > maxCacheSize) {
            // Cache is full
            return false;
        } else if (maxSegmentSize - localOffset < size) {
            // If an entry is at the end of a segment, we need to get a new offset and try
            // again in next segment
            continue;
        } else {
            // Found a good offset
            break;
        }
    }

    cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());

    // Update last entryId for ledger. This logic is to handle writes for the same
    // ledger coming out of order and from different thread, though in practice it
    // should not happen and the compareAndSet should be always uncontended.
    while (true) {
        long currentLastEntryId = lastEntryMap.get(ledgerId);
        if (currentLastEntryId > entryId) {
            // A newer entry is already there
            break;
        }

        if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
            break;
        }
    }

    index.put(ledgerId, entryId, offset, size);
    cacheCount.increment();
    cacheSize.addAndGet(size);
    return true;
}

```

Figura 17

## WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.
• <a href="#">forEach(WriteCache.EntryConsumer)</a>		0%		0%
• <a href="#">lambda\$foreach\$0(long, long, long, long)</a>		0%		0%
• <a href="#">getLastEntry(long)</a>		0%		0%
• <a href="#">isEmpty()</a>		0%		0%
• <a href="#">deleteLedger(long)</a>		0%		n/a
• <a href="#">size()</a>		0%		n/a
• <a href="#">count()</a>		0%		n/a
• <a href="#">WriteCache(ByteBufAllocator, long, int)</a>		98%		66%
• <a href="#">put(long, long, ByteBuf)</a>		97%		75%
• <a href="#">get(long, long)</a>		100%		100%
• <a href="#">clear()</a>		100%		n/a
• <a href="#">close()</a>		100%		100%
• <a href="#">alignToPowerOfTwo(long)</a>		100%		n/a
• <a href="#">static {...}</a>		100%		n/a
• <a href="#">align64(int)</a>		100%		n/a
• <a href="#">WriteCache(ByteBufAllocator, long)</a>		100%		n/a
Total	264 of 617	57%	24 of 38	36%

Figura 18

```
if ((offset + size) > maxCacheSize) {
    // Cache is full
    return false;
} else if (maxSegmentsize - localOffset < size) {
    // If an entry is at the end of a segment, we need
    // again in next segment
    continue;
} else {
    // Found a good offset
    break;
}
```

Figura 19

```

<method name="put" desc="(JJLio/netty/buffer/ByteBuf;)Z">
<du var="this" def="132" use="143" covered="1"/>
<du var="this" def="132" use="144" covered="1"/>
<du var="this" def="132" use="145" covered="1"/>
<du var="this" def="132" use="147" target="149" covered="1"/>
<du var="this" def="132" use="147" target="150" covered="1"/>
<du var="this" def="132" use="150" target="153" covered="1"/>
<du var="this" def="132" use="150" target="160" covered="1"/>
<du var="this" def="132" use="160" covered="1"/>
<du var="this" def="132" use="166" covered="1"/>
<du var="this" def="132" use="172" target="173" covered="1"/>
<du var="this" def="132" use="172" target="175" covered="0"/>
<du var="this" def="132" use="177" covered="1"/>
<du var="this" def="132" use="178" covered="1"/>
<du var="this" def="132" use="179" covered="1"/>
<du var="ledgerId" def="132" use="166" covered="1"/>
<du var="ledgerId" def="132" use="172" target="173" covered="1"/>
<du var="ledgerId" def="132" use="172" target="175" covered="0"/>
<du var="ledgerId" def="132" use="177" covered="1"/>
<du var="entryId" def="132" use="167" target="169" covered="0"/>
<du var="entryId" def="132" use="167" target="172" covered="1"/>
<du var="entryId" def="132" use="172" target="173" covered="1"/>
<du var="entryId" def="132" use="172" target="175" covered="0"/>
<du var="entryId" def="132" use="177" covered="1"/>
<du var="entry" def="132" use="160" covered="1"/>
<du var="this.cacheOffset" def="132" use="143" covered="1"/>
<du var="this.segmentOffsetMask" def="132" use="144" covered="1"/>
<du var="this.segmentOffsetBits" def="132" use="145" covered="1"/>
<du var="this.maxCacheSize" def="132" use="147" target="149" covered="1"/>
<du var="this.maxCacheSize" def="132" use="147" target="150" covered="1"/>
<du var="this.maxSegmentsize" def="132" use="150" target="153" covered="1"/>
<du var="this.maxSegmentsize" def="132" use="150" target="160" covered="1"/>
<du var="this.cacheSegments" def="132" use="160" covered="1"/>
<du var="this.lastEntryMap" def="132" use="166" covered="1"/>
<du var="this.lastEntryMap" def="132" use="172" target="173" covered="1"/>
<du var="this.lastEntryMap" def="132" use="172" target="175" covered="0"/>
<du var="this.index" def="132" use="177" covered="1"/>
<du var="this.cacheCount" def="132" use="178" covered="1"/>
<du var="this.cacheSize" def="132" use="179" covered="1"/>
<du var="size" def="132" use="147" target="149" covered="1"/>
<du var="size" def="132" use="147" target="150" covered="1"/>
<du var="size" def="132" use="150" target="153" covered="1"/>
<du var="size" def="132" use="150" target="160" covered="1"/>
<du var="size" def="132" use="177" covered="1"/>
<du var="size" def="132" use="179" covered="1"/>
<du var="alignedSize" def="136" use="143" covered="1"/>
<du var="offset" def="143" use="147" target="149" covered="1"/>
<du var="offset" def="143" use="147" target="150" covered="1"/>
<du var="offset" def="143" use="177" covered="1"/>
<du var="localoffset" def="144" use="150" target="153" covered="1"/>
<du var="localoffset" def="144" use="150" target="160" covered="1"/>
<du var="localoffset" def="144" use="160" covered="1"/>
<du var="segmentIdx" def="145" use="160" covered="1"/>
<du var="currentLastEntryId" def="166" use="167" target="169" covered="0"/>
<du var="currentLastEntryId" def="166" use="167" target="172" covered="1"/>
<du var="currentLastEntryId" def="166" use="172" target="173" covered="1"/>
<du var="currentLastEntryId" def="166" use="172" target="175" covered="0"/>
<counter type="DU" missed="7" covered="49"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 20

```

<method name="get" desc="(JJ)Lio/netty/buffer/ByteBuf;">
<du var="this" def="184" use="191" covered="1"/>
<du var="this" def="184" use="193" covered="1"/>
<du var="this" def="184" use="194" covered="1"/>
<du var="this" def="184" use="195" covered="1"/>
<du var="thisallocator" def="184" use="191" covered="1"/>
<du var="this.segmentOffsetMask" def="184" use="193" covered="1"/>
<du var="this.segmentOffsetBits" def="184" use="194" covered="1"/>
<du var="this.cacheSegments" def="184" use="195" covered="1"/>
<du var="result" def="184" use="185" target="186" covered="1"/>
<du var="result" def="184" use="185" target="189" covered="1"/>
<du var="result" def="184" use="189" covered="1"/>
<du var="result" def="184" use="190" covered="1"/>
<du var="result.first" def="184" use="189" covered="1"/>
<du var="result.second" def="184" use="190" covered="1"/>
<counter type="DU" missed="0" covered="14"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 21

```

public ByteBuf get(long ledgerId, long entryId) {
    LongPair result = index.get(ledgerId, entryId);
    if (result == null) {
        return null;
    }

    long offset = result.first;
    int size = (int) result.second;
    ByteBuf entry = allocator.buffer(size, size);

    int localOffset = (int) (offset & segmentOffsetMask);
    int segmentIdx = (int) (offset >>> segmentOffsetBits);
    entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
    return entry;
}

```

Figura 22

## CoordinatedBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.
checkFinishId(Tuple, CoordinatedBolt.TupleType)	29%	26%		
prepare(Map, TopologyContext, OutputCollector)	73%	66%		
execute(Tuple)	70%	75%		
declareOutputFields(OutputFieldsDeclarer)	0%	n/a		
getTupleType(Tuple)	72%	50%		
getComponentConfiguration()	0%	n/a		
CoordinatedBolt(IRichBolt, Map, CoordinatedBolt.IdStreamSpec)	100%	100%		
singleSourceArgs(String, CoordinatedBolt.SourceArgs)	100%	n/a		
CoordinatedBolt(IRichBolt, String, CoordinatedBolt.SourceArgs, CoordinatedBolt.IdStreamSpec)	100%	n/a		
cleanup()	100%	n/a		
CoordinatedBolt(IRichBolt)	100%	n/a		
static {...}	100%	n/a		
Total	260 of 573	54%	41 of 72	43%

Figura 23

```

@Override
public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
    TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
    if (delegate instanceof TimeoutCallback) {
        callback = new TimeoutItems();
    }
    tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
    this.collector = collector;
    delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
    for (String component : Utils.get(context.getThisTargets(),
        Constants.COORDINATED_STREAM_ID,
        new HashMap<String, Grouping>()))
        .keySet()) {
            for (Integer task : context.getComponentTasks(component)) {
                countOutTasks.add(task);
            }
        }
    if (!sourceArgs.isEmpty()) {
        numSourceReports = 0;
        for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
            if (entry.getValue().singleCount) {
                numSourceReports += 1;
            } else {
                numSourceReports += context.getComponentTasks(entry.getKey()).size();
            }
        }
    }
}

```

Figura 24

```

<method name="prepare" desc="(Ljava/util/Map;Lorg/apache/storm/task/Topo
<du var="this" def="79" use="80" target="81" covered="1"/>
<du var="this" def="79" use="80" target="83" covered="1"/>
<du var="this" def="79" use="83" covered="1"/>
<du var="this" def="79" use="84" covered="1"/>
<du var="this" def="79" use="85" covered="1"/>
<du var="this" def="79" use="85" covered="1"/>
<du var="this" def="79" use="94" target="95" covered="1"/>
<du var="this" def="79" use="94" target="104" covered="1"/>
<du var="this" def="79" use="95" covered="1"/>
<du var="this" def="79" use="96" covered="1"/>
<du var="this" def="79" use="100" covered="1"/>
<du var="this" def="79" use="98" covered="0"/>
<du var="this" def="79" use="91" covered="0"/>
<du var="this" def="79" use="81" covered="1"/>
<du var="config" def="79" use="85" covered="1"/>
<du var="context" def="79" use="83" covered="1"/>
<du var="context" def="79" use="85" covered="1"/>
<du var="context" def="79" use="86" covered="1"/>
<du var="context" def="79" use="100" covered="1"/>
<du var="context" def="79" use="90" covered="0"/>
<du var="collector" def="79" use="84" covered="1"/>
<du var="collector" def="79" use="85" covered="1"/>
<du var="this.delegate" def="79" use="80" target="81" covered="1"/>
<du var="this.delegate" def="79" use="80" target="83" covered="1"/>
<du var="this.delegate" def="79" use="85" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="79" use="86" covered="1"/>
<du var="this.countOutTasks" def="79" use="91" covered="0"/>
<du var="this.sourceArgs" def="79" use="94" target="95" covered="1"/>
<du var="this.sourceArgs" def="79" use="94" target="104" covered="1"/>
<du var="this.sourceArgs" def="79" use="96" covered="1"/>
<du var="callback" def="79" use="83" covered="1"/>
<du var="callback" def="81" use="83" covered="1"/>
<du var="this.numSourceReports" def="95" use="100" covered="1"/>
<du var="this.numSourceReports" def="95" use="98" covered="0"/>
<du var="entry" def="96" use="97" target="98" covered="0"/>
<du var="entry" def="96" use="97" target="100" covered="1"/>
<du var="entry" def="96" use="100" covered="1"/>
<du var="this.numSourceReports" def="98" use="100" covered="0"/>
<du var="this.numSourceReports" def="98" use="98" covered="0"/>
<du var="this.numSourceReports" def="100" use="100" covered="0"/>
<du var="this.numSourceReports" def="100" use="98" covered="0"/>
<counter type="DU" missed="15" covered="35"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 25

```

@Override
public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
    TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
    if (delegate instanceof TimeoutCallback) {
        callback = new TimeoutItems();
    }
    tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
    this.collector = collector;
    delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
    for (String component : Utils.get(context.getThisTargets(),
            Constants.COORDINATED_STREAM_ID,
            new HashMap<String, Grouping>())
            .keySet()) {
        for (Integer task : context.getComponentTasks(component)) {
            countOutTasks.add(task);
        }
    }
    if (!sourceArgs.isEmpty()) {
        numSourceReports = 0;
        for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
            if (entry.getValue().singleCount) {
                numSourceReports += 1;
            } else {
                numSourceReports += context.getComponentTasks(entry.getKey()).size();
            }
        }
    }
}

```

Figura 26

## CoordinatedBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.
checkFinishId(Tuple, CoordinatedBolt.TupleType)	29%	26%		
execute(Tuple)	70%	75%		
prepare(Map, TopologyContext, OutputCollector)	81%	75%		
declareOutputFields(OutputFieldsDeclarer)	0%	n/a		
getComponentConfiguration()	0%	n/a		
getTupleType(Tuple)	92%	75%		
CoordinatedBolt(IRichBolt, Map, CoordinatedBolt.IdStreamSpec)	100%	100%		
singleSourceArgs(String, CoordinatedBolt.SourceArgs)	100%	n/a		
CoordinatedBolt(IRichBolt, String, CoordinatedBolt.SourceArgs, CoordinatedBolt.IdStreamSpec)	100%	n/a		
cleanup()	100%	n/a		
CoordinatedBolt(IRichBolt)	100%	n/a		
static {...}	100%	n/a		
Total	246 of 573	57%	38 of 72	47%

Figura 27

```

@Override
public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
    TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
    if (delegate instanceof TimeoutCallback) {
        callback = new TimeoutItems();
    }
    tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
    this.collector = collector;
    delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
    for (String component : Utils.get(context.getThisTargets(),
        Constants.COORDINATED_STREAM_ID,
        new HashMap<String, Grouping>()))
        .keySet()) {
        for (Integer task : context.getComponentTasks(component)) {
            countOutTasks.add(task);
        }
    }
    if (!sourceArgs.isEmpty()) {
        numSourceReports = 0;
        for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
            if (entry.getValue().singleCount) {
                numSourceReports += 1;
            } else {
                numSourceReports += context.getComponentTasks(entry.getKey()).size();
            }
        }
    }
}

```

Figura 28

```

<method name="prepare" desc="(Ljava/util/Map;Lorg/apache/storm/task/Topo
<du var="this" def="79" use="80" target="81" covered="1"/>
<du var="this" def="79" use="80" target="83" covered="1"/>
<du var="this" def="79" use="83" covered="1"/>
<du var="this" def="79" use="84" covered="1"/>
<du var="this" def="79" use="85" covered="1"/>
<du var="this" def="79" use="85" covered="1"/>
<du var="this" def="79" use="94" target="95" covered="1"/>
<du var="this" def="79" use="94" target="104" covered="1"/>
<du var="this" def="79" use="95" covered="1"/>
<du var="this" def="79" use="96" covered="1"/>
<du var="this" def="79" use="100" covered="1"/>
<du var="this" def="79" use="98" covered="1"/>
<du var="this" def="79" use="91" covered="0"/>
<du var="this" def="79" use="81" covered="1"/>
<du var="config" def="79" use="85" covered="1"/>
<du var="context" def="79" use="83" covered="1"/>
<du var="context" def="79" use="85" covered="1"/>
<du var="context" def="79" use="86" covered="1"/>
<du var="context" def="79" use="100" covered="1"/>
<du var="context" def="79" use="90" covered="0"/>
<du var="collector" def="79" use="84" covered="1"/>
<du var="collector" def="79" use="85" covered="1"/>
<du var="this.delegate" def="79" use="80" target="81" covered="1"/>
<du var="this.delegate" def="79" use="80" target="83" covered="1"/>
<du var="this.delegate" def="79" use="85" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="79" use="86" covered="1"/>
<du var="this.countOutTasks" def="79" use="91" covered="0"/>
<du var="this.sourceArgs" def="79" use="94" target="95" covered="1"/>
<du var="this.sourceArgs" def="79" use="94" target="104" covered="1"/>
<du var="this.sourceArgs" def="79" use="96" covered="1"/>
<du var="callback" def="79" use="83" covered="1"/>
<du var="callback" def="81" use="83" covered="1"/>
<du var="this.numSourceReports" def="95" use="100" covered="1"/>
<du var="this.numSourceReports" def="95" use="98" covered="1"/>
<du var="entry" def="96" use="97" target="98" covered="1"/>
<du var="entry" def="96" use="97" target="100" covered="1"/>
<du var="entry" def="96" use="100" covered="1"/>
<du var="this.numSourceReports" def="98" use="100" covered="0"/>
<du var="this.numSourceReports" def="98" use="98" covered="0"/>
<du var="this.numSourceReports" def="100" use="100" covered="0"/>
<du var="this.numSourceReports" def="100" use="98" covered="0"/>
<counter type="DU" missed="12" covered="38"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 29

```

@Override
public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
    TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
    if (delegate instanceof TimeoutCallback) {
        callback = new TimeoutItems();
    }
    tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
    this.collector = collector;
    delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
    for (String component : Utils.get(context.getThisTargets(),
        Constants.COORDINATED_STREAM_ID,
        new HashMap<String, Grouping>())
        .keySet()) {
        for (Integer task : context.getComponentTasks(component)) {
            countOutTasks.add(task);
        }
    }
    if (!sourceArgs.isEmpty()) {
        numSourceReports = 0;
        for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
            if (entry.getValue().singleCount) {
                numSourceReports += 1;
            } else {
                numSourceReports += context.getComponentTasks(entry.getKey()).size();
            }
        }
    }
}

```

Figura 30

```

@Override
public void execute(Tuple tuple) {
    Object id = tuple.getValue(0);
    TrackingInfo track;
    TupleType type = getTupleType(tuple);
    synchronized (tracked) {
        track = tracked.get(id);
        if (track == null) {
            track = new TrackingInfo();
            if (idStreamSpec == null) {
                track.receivedId = true;
            }
        }
        tracked.put(id, track);
    }
}

if (type == TupleType.ID) {
    synchronized (tracked) {
        track.receivedId = true;
    }
    checkFinishId(tuple, type);
} else if (type == TupleType.COORD) {
    int count = (Integer) tuple.getValue(1);
    synchronized (tracked) {
        track.reportCount++;
        track.expectedTupleCount += count;
    }
    checkFinishId(tuple, type);
} else {
    synchronized (tracked) {
        delegate.execute(tuple);
    }
}

```

Figura 31

```

<method name="execute" desc="(Lorg/apache/storm/tuple/Tuple;)V">
<du var="this" def="171" use="198" covered="1"/>
<du var="this" def="171" use="198" covered="1"/>
<du var="this" def="171" use="199" covered="1"/>
<du var="this" def="171" use="192" covered="0"/>
<du var="this" def="171" use="192" covered="0"/>
<du var="this" def="171" use="196" covered="0"/>
<du var="this" def="171" use="186" covered="1"/>
<du var="this" def="171" use="186" covered="1"/>
<du var="this" def="171" use="189" covered="1"/>
<du var="this" def="171" use="178" target="179" covered="1"/>
<du var="this" def="171" use="178" target="181" covered="1"/>
<du var="this" def="171" use="181" covered="1"/>
<du var="tuple" def="171" use="199" covered="1"/>
<du var="tuple" def="171" use="191" covered="0"/>
<du var="tuple" def="171" use="196" covered="0"/>
<du var="tuple" def="171" use="189" covered="1"/>
<du var="this.tracked" def="171" use="198" covered="1"/>
<du var="this.tracked" def="171" use="198" covered="1"/>
<du var="this.tracked" def="171" use="192" covered="0"/>
<du var="this.tracked" def="171" use="192" covered="0"/>
<du var="this.tracked" def="171" use="186" covered="1"/>
<du var="this.tracked" def="171" use="186" covered="1"/>
<du var="this.tracked" def="171" use="181" covered="1"/>
<du var="this.idStreamSpec" def="171" use="178" target="179" covered="1"/>
<du var="this.idStreamSpec" def="171" use="178" target="181" covered="1"/>
<du var="ID" def="171" use="185" target="186" covered="1"/>
<du var="ID" def="171" use="185" target="190" covered="1"/>
<du var="COORD" def="171" use="190" target="191" covered="0"/>
<du var="COORD" def="171" use="190" target="198" covered="1"/>
<du var="this.delegate" def="171" use="199" covered="1"/>
<du var="id" def="171" use="181" covered="1"/>
<du var="type" def="173" use="185" target="186" covered="1"/>
<du var="type" def="173" use="185" target="190" covered="1"/>
<du var="type" def="173" use="190" target="191" covered="0"/>
<du var="type" def="173" use="190" target="198" covered="1"/>
<du var="type" def="173" use="196" covered="0"/>
<du var="type" def="173" use="189" covered="1"/>
<du var="track" def="175" use="176" target="177" covered="1"/>
<du var="track" def="175" use="176" target="183" covered="0"/>
<du var="track" def="175" use="193" covered="0"/>
<du var="track" def="175" use="194" covered="0"/>
<du var="track" def="175" use="187" covered="0"/>
<du var="track.reportCount" def="175" use="193" covered="0"/>
<du var="track.expectedTupleCount" def="175" use="194" covered="0"/>
<du var="track" def="177" use="181" covered="1"/>
<du var="track" def="177" use="193" covered="0"/>
<du var="track" def="177" use="194" covered="0"/>
<du var="track" def="177" use="187" covered="1"/>
<du var="track" def="177" use="179" covered="1"/>
<du var="track.reportCount" def="177" use="193" covered="0"/>
<du var="track.expectedTupleCount" def="177" use="194" covered="0"/>
<counter type="DU" missed="20" covered="32"/>

```

Figura 32

```

@Override
public void execute(Tuple tuple) {
    Object id = tuple.getValue(0);
    TrackingInfo track;
    TupleType type = getTupleType(tuple);
    synchronized (tracked) {
        track = tracked.get(id);
        if (track == null) {
            track = new TrackingInfo();
            if (idStreamSpec == null) {
                track.receivedId = true;
            }
            tracked.put(id, track);
        }
    }

    if (type == TupleType.ID) {
        synchronized (tracked) {
            track.receivedId = true;
        }
        checkFinishId(tuple, type);
    } else if (type == TupleType.COORD) {
        int count = (Integer) tuple.getValue(1);
        synchronized (tracked) {
            track.reportCount++;
            track.expectedTupleCount += count;
        }
        checkFinishId(tuple, type);
    } else {
        synchronized (tracked) {
            delegate.execute(tuple);
        }
    }
}

```

Figura 33

## CoordinatedBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.
checkFinishId(Tuple, CoordinatedBolt.TupleType)	31%	30%		
prepare(Map, TopologyContext, OutputCollector)	81%	75%		
declareOutputFields(OutputFieldsDeclarer)	0%	n/a		
getComponentConfiguration()	0%	n/a		
execute(Tuple)	100%	87%		
CoordinatedBolt(IRichBolt, Map, CoordinatedBolt.IdStreamSpec)	100%	100%		
getTupleType(Tuple)	100%	87%		
singleSourceArgs(String, CoordinatedBolt.SourceArgs)	100%	n/a		
CoordinatedBolt(IRichBolt, String, CoordinatedBolt.SourceArgs, CoordinatedBolt.IdStreamSpec)	100%	n/a		
cleanup()	100%	n/a		
CoordinatedBolt(IRichBolt)	100%	n/a		
static {...}	100%	n/a		
Total	209 of 573	63%	34 of 72	52%

Figura 34

```

@Override
public void execute(Tuple tuple) {
    Object id = tuple.getValue(0);
    TrackingInfo track;
    TupleType type = getTupleType(tuple);
    synchronized (tracked) {
        track = tracked.get(id);
        if (track == null) {
            track = new TrackingInfo();
            if (idStreamSpec == null) {
                track.receivedId = true;
            }
        }
        tracked.put(id, track);
    }

    if (type == TupleType.ID) {
        synchronized (tracked) {
            track.receivedId = true;
        }
        checkFinishId(tuple, type);
    } else if (type == TupleType.COORD) {
        int count = (Integer) tuple.getValue(1);
        synchronized (tracked) {
            track.reportCount++;
            track.expectedTupleCount += count;
        }
        checkFinishId(tuple, type);
    } else {
        synchronized (tracked) {
            delegate.execute(tuple);
        }
    }
}

```

Figura 35

```

<method name="execute" desc="(Lorg/apache/storm/tuple/Tuple;)V">
<du var="this" def="171" use="198" covered="1"/>
<du var="this" def="171" use="198" covered="1"/>
<du var="this" def="171" use="199" covered="1"/>
<du var="this" def="171" use="192" covered="1"/>
<du var="this" def="171" use="192" covered="1"/>
<du var="this" def="171" use="196" covered="1"/>
<du var="this" def="171" use="186" covered="1"/>
<du var="this" def="171" use="186" covered="1"/>
<du var="this" def="171" use="189" covered="1"/>
<du var="this" def="171" use="178" target="179" covered="1"/>
<du var="this" def="171" use="178" target="181" covered="1"/>
<du var="this" def="171" use="181" covered="1"/>
<du var="tuple" def="171" use="199" covered="1"/>
<du var="tuple" def="171" use="191" covered="1"/>
<du var="tuple" def="171" use="196" covered="1"/>
<du var="tuple" def="171" use="189" covered="1"/>
<du var="this.tracked" def="171" use="198" covered="1"/>
<du var="this.tracked" def="171" use="198" covered="1"/>
<du var="this.tracked" def="171" use="192" covered="1"/>
<du var="this.tracked" def="171" use="192" covered="1"/>
<du var="this.tracked" def="171" use="186" covered="1"/>
<du var="this.tracked" def="171" use="186" covered="1"/>
<du var="this.tracked" def="171" use="181" covered="1"/>
<du var="this.idStreamSpec" def="171" use="178" target="179" covered="1"/>
<du var="this.idStreamSpec" def="171" use="178" target="181" covered="1"/>
<du var="ID" def="171" use="185" target="186" covered="1"/>
<du var="ID" def="171" use="185" target="190" covered="1"/>
<du var="COORD" def="171" use="190" target="191" covered="1"/>
<du var="COORD" def="171" use="190" target="198" covered="1"/>
<du var="this.delegate" def="171" use="199" covered="1"/>
<du var="id" def="171" use="181" covered="1"/>
<du var="type" def="173" use="185" target="186" covered="1"/>
<du var="type" def="173" use="185" target="190" covered="1"/>
<du var="type" def="173" use="190" target="191" covered="1"/>
<du var="type" def="173" use="190" target="198" covered="1"/>
<du var="type" def="173" use="196" covered="1"/>
<du var="type" def="173" use="189" covered="1"/>
<du var="track" def="175" use="176" target="177" covered="1"/>
<du var="track" def="175" use="176" target="183" covered="0"/>
<du var="track" def="175" use="193" covered="0"/>
<du var="track" def="175" use="194" covered="0"/>
<du var="track" def="175" use="187" covered="0"/>
<du var="track.reportCount" def="175" use="193" covered="0"/>
<du var="track.expectedTupleCount" def="175" use="194" covered="0"/>
<du var="track" def="177" use="181" covered="1"/>
<du var="track" def="177" use="193" covered="1"/>
<du var="track" def="177" use="194" covered="1"/>
<du var="track" def="177" use="187" covered="1"/>
<du var="track" def="177" use="179" covered="1"/>
<du var="track.reportCount" def="177" use="193" covered="1"/>
<du var="track.expectedTupleCount" def="177" use="194" covered="1"/>
<counter type="DU" missed="6" covered="46"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 36

```

@Override
public void execute(Tuple tuple) {
    Object id = tuple.getValue(0);
    TrackingInfo track;
    TupleType type = getTupleType(tuple);
    synchronized (tracked) {
        track = tracked.get(id);
        if (track == null) {
            track = new TrackingInfo();
            if (idStreamSpec == null) {
                track.receivedId = true;
            }
            tracked.put(id, track);
        }
    }

    if (type == TupleType.ID) {
        synchronized (tracked) {
            track.receivedId = true;
        }
        checkFinishId(tuple, type);
    } else if (type == TupleType.COORD) {
        int count = (Integer) tuple.getValue(1);
        synchronized (tracked) {
            track.reportCount++;
            track.expectedTupleCount += count;
        }
        checkFinishId(tuple, type);
    } else {
        synchronized (tracked) {
            delegate.execute(tuple);
        }
    }
}

```

Figura 37

```

private TupleType getTupleType(Tuple tuple) {
    if (idStreamSpec != null
        && tuple.getSourceGlobalStreamId().equals(idStreamSpec.id)) {
        return TupleType.ID;
    } else if (!sourceArgs.isEmpty()
        && tuple.getSourceStreamId().equals(Constants.COORDINATED_STREAM_ID)) {
        return TupleType.COORD;
    } else {
        return TupleType.REGULAR;
    }
}

enum TupleType {
    REGULAR,
    ID,
    COORD
}

```

Figura 38

```

private TupleType getTupleType(Tuple tuple) {
    if (idStreamSpec != null
        && tuple.getSourceGlobalStreamId().equals(idStreamSpec.id)) {
        return TupleType.ID;
    } else if (!sourceArgs.isEmpty()
        && tuple.getSourceStreamId().equals(Constants.COORDINATED_STREAM_ID)) {
        return TupleType.COORD;
    } else {
        return TupleType.REGULAR;
    }
}

enum TupleType {
    REGULAR,
    ID,
    COORD
}

```

Figura 39

```

<method name="getTupleType" desc="(Lorg/apache/storm/tuple/Tuple;)Lorg/apache/sto
<du var="this" def="222" use="222" target="222" covered="1"/>
<du var="this" def="222" use="222" target="225" covered="1"/>
<du var="this" def="222" use="225" target="225" covered="0"/>
<du var="this" def="222" use="225" target="229" covered="1"/>
<du var="this" def="222" use="223" target="224" covered="1"/>
<du var="this" def="222" use="223" target="225" covered="0"/>
<du var="tuple" def="222" use="226" target="227" covered="0"/>
<du var="tuple" def="222" use="226" target="229" covered="0"/>
<du var="tuple" def="222" use="223" target="224" covered="1"/>
<du var="tuple" def="222" use="223" target="225" covered="0"/>
<du var="this.idStreamSpec" def="222" use="222" target="222" covered="1"/>
<du var="this.idStreamSpec" def="222" use="222" target="225" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="224" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="225" covered="0"/>
<du var="this.id" def="222" use="223" target="224" covered="1"/>
<du var="this.id" def="222" use="223" target="225" covered="0"/>
<du var="ID" def="222" use="224" covered="1"/>
<du var="this.sourceArgs" def="222" use="225" target="225" covered="0"/>
<du var="this.sourceArgs" def="222" use="225" target="229" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="227" covered="0"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="229" covered="0"/>
<du var="COORD" def="222" use="227" covered="0"/>
<du var="REGULAR" def="222" use="229" covered="1"/>
<counter type="DU" missed="11" covered="12"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 40

```

<method name="getTupleType" desc="(Lorg/apache/storm/tuple/Tuple;)Lorg/apache/st
<du var="this" def="222" use="222" target="222" covered="1"/>
<du var="this" def="222" use="222" target="225" covered="1"/>
<du var="this" def="222" use="225" target="225" covered="1"/>
<du var="this" def="222" use="225" target="229" covered="1"/>
<du var="this" def="222" use="223" target="224" covered="1"/>
<du var="this" def="222" use="223" target="225" covered="0"/>
<du var="tuple" def="222" use="226" target="227" covered="0"/>
<du var="tuple" def="222" use="226" target="229" covered="1"/>
<du var="tuple" def="222" use="223" target="224" covered="1"/>
<du var="tuple" def="222" use="223" target="225" covered="0"/>
<du var="this.idStreamSpec" def="222" use="222" target="222" covered="1"/>
<du var="this.idStreamSpec" def="222" use="222" target="225" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="224" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="225" covered="0"/>
<du var="this.id" def="222" use="223" target="224" covered="1"/>
<du var="this.id" def="222" use="223" target="225" covered="0"/>
<du var="ID" def="222" use="224" covered="1"/>
<du var="this.sourceArgs" def="222" use="225" target="225" covered="1"/>
<du var="this.sourceArgs" def="222" use="225" target="229" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="227" covered="0"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="229" covered="1"/>
<du var="COORD" def="222" use="227" covered="0"/>
<du var="REGULAR" def="222" use="229" covered="1"/>
<counter type="DU" missed="7" covered="16"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 41

```

<method name="getTupleType" desc="(Lorg/apache/storm/tuple/Tuple;)Lorg/apache/st
<du var="this" def="222" use="222" target="222" covered="1"/>
<du var="this" def="222" use="222" target="225" covered="1"/>
<du var="this" def="222" use="225" target="225" covered="1"/>
<du var="this" def="222" use="225" target="229" covered="1"/>
<du var="this" def="222" use="223" target="224" covered="1"/>
<du var="this" def="222" use="223" target="225" covered="0"/>
<du var="tuple" def="222" use="226" target="227" covered="1"/>
<du var="tuple" def="222" use="226" target="229" covered="1"/>
<du var="tuple" def="222" use="223" target="224" covered="1"/>
<du var="tuple" def="222" use="223" target="225" covered="0"/>
<du var="this.idStreamSpec" def="222" use="222" target="222" covered="1"/>
<du var="this.idStreamSpec" def="222" use="222" target="225" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="224" covered="1"/>
<du var="this.idStreamSpec" def="222" use="223" target="225" covered="0"/>
<du var="this.id" def="222" use="223" target="224" covered="1"/>
<du var="this.id" def="222" use="223" target="225" covered="0"/>
<du var="ID" def="222" use="224" covered="1"/>
<du var="this.sourceArgs" def="222" use="225" target="225" covered="1"/>
<du var="this.sourceArgs" def="222" use="225" target="229" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="227" covered="1"/>
<du var="COORDINATED_STREAM_ID" def="222" use="226" target="229" covered="1"/>
<du var="COORD" def="222" use="227" covered="1"/>
<du var="REGULAR" def="222" use="229" covered="1"/>
<counter type="DU" missed="4" covered="19"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 42

```

// 1-5
inputs.add(new TestInput("", "streamTest", -1, objectList1, CoordinatedBolt.SourceArgs.all(),
CoordinatedBolt.IdStreamSpec.makeDetectSpec("", "streamTest")));

inputs.add(new TestInput("srcTest", "streamTest", 10, objectList1, CoordinatedBolt.SourceArgs.all(),
CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest", "streamTest")));

inputs.add(new TestInput("srcTest", "", -1, objectList1, CoordinatedBolt.SourceArgs.all(),
CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest", "")));

inputs.add(new TestInput("srcTest", Constants.COORDINATED_STREAM_ID, 0, objectList1,
CoordinatedBolt.SourceArgs.all(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest",
Constants.COORDINATED_STREAM_ID)));

inputs.add(new TestInput("", Constants.COORDINATED_STREAM_ID, 10, objectList1,
CoordinatedBolt.SourceArgs.all(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("", 
Constants.COORDINATED_STREAM_ID)));

// 6-10
inputs.add(new TestInput("", Constants.COORDINATED_STREAM_ID, 10, objectList2,
CoordinatedBolt.SourceArgs.single(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("", 
Constants.COORDINATED_STREAM_ID)));

inputs.add(new TestInput(null, "streamTest", 0, objectList2, CoordinatedBolt.SourceArgs.single(),
null));

inputs.add(new TestInput("srcTest", "", -1, objectList2, CoordinatedBolt.SourceArgs.single(),
CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest", "")));

inputs.add(new TestInput(null, "", -1, objectList2, CoordinatedBolt.SourceArgs.all(), null));

inputs.add(new TestInput("srcTest", "streamTest", null, objectList2,
CoordinatedBolt.SourceArgs.single(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest",
"streamTest")));

// 11-15
inputs.add(new TestInput("srcTest", "streamTest", 0, objectList1,
CoordinatedBolt.SourceArgs.single(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest",
"streamTest")));

inputs.add(new TestInput("srcTest", "streamTest", -1, objectList1,
CoordinatedBolt.SourceArgs.single(), CoordinatedBolt.IdStreamSpec.makeDetectSpec("srcTest",
"streamTest")));

inputs.add(new TestInput("srcTest", "streamTest", 10, objectList2, CoordinatedBolt.SourceArgs.all(),
null));

inputs.add(new TestInput("srcTest", Constants.COORDINATED_STREAM_ID, 10, objectList2,
CoordinatedBolt.SourceArgs.all(), null));

inputs.add(new TestInput("srcTest", "", null, objectList2, CoordinatedBolt.SourceArgs.all(), null));

```

Figura 43

```

private TupleType getTupleType(Tuple tuple) {
    if (idStreamSpec != null
        && tuple.getSourceGlobalStreamId().equals(idStreamSpec.id)) {
        return TupleType.ID;
    } else if (!sourceArgs.isEmpty()
        && tuple.getSourceStreamId().equals(Constants.COORDINATED_STREAM_ID)) {
        return TupleType.COORD;
    } else {
        return TupleType.REGULAR;
    }
}

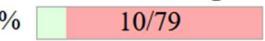
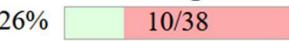
```

Figura 44

# Pit Test Coverage Report

## Package Summary

### org.apache.storm.coordination

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	53%  106/200	13%  10/79	26%  10/38

## Breakdown by Class

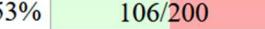
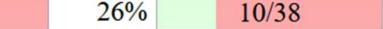
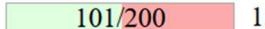
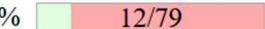
Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">CoordinatedBolt.java</a>	53%  106/200	13%  10/79	26%  10/38

Figura 45

# Pit Test Coverage Report

## Package Summary

### org.apache.storm.coordination

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	51%  101/200	15%  12/79	32%  12/37

## Breakdown by Class

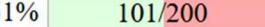
Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">CoordinatedBolt.java</a>	51%  101/200	15%  12/79	32%  12/37

Figura 46

## JoinBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
declareOutputFields(OutputFieldsDeclarer)	0%	0%	0%	0%	3	3	7	7	1	1
doLeftJoin(JoinBolt.JoinAccumulator_Map_JoinBolt.JoinInfo_boolean)	67%	67%	40%	40%	4	6	4	16	0	1
doInnerJoin(JoinBolt.JoinAccumulator_Map_JoinBolt.JoinInfo_boolean)	66%	66%	50%	50%	3	5	4	14	0	1
getStreamSelector(Tuple)	36%	36%	2%	33%	2	3	2	4	0	1
lookupField(JoinBolt.FieldSelector_Tuple)	75%	75%	66%	66%	3	7	4	13	0	1
doJoin(JoinBolt.JoinAccumulator_HashMap_JoinBolt.JoinInfo_boolean)	62%	62%	66%	66%	1	3	1	5	0	1
execute(TupleWindow)	79%	79%	75%	75%	1	3	1	9	0	1
JoinBolt(String_String)	0%	0%	n/a	n/a	1	1	2	2	1	1
withWindow(BaseWindowedBolt_Count_BaseWindowedBolt_Count)	0%	0%	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt_Count_BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt_Duration_BaseWindowedBolt_Count)	0%	0%	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt_Duration_BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
prepare(Map_TopologyContext_OutputCollector)	86%	86%	83%	83%	1	4	1	10	0	1
withOutputStream(String)	0%	0%	n/a	n/a	1	1	2	2	1	1
withWindow(BaseWindowedBolt_Count)	0%	0%	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
withTumblingWindow(BaseWindowedBolt_Count)	0%	0%	n/a	n/a	1	1	1	1	1	1
withTumblingWindow(BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
withTimestampField(String)	0%	0%	n/a	n/a	1	1	1	1	1	1
withTimestampExtractor(TimestampExtractor)	0%	0%	n/a	n/a	1	1	1	1	1	1
withLateTupleStream(String)	0%	0%	n/a	n/a	1	1	1	1	1	1
withLag(BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
withWatermarkInterval(BaseWindowedBolt_Duration)	0%	0%	n/a	n/a	1	1	1	1	1	1
hashJoin(List)	100%	100%	100%	100%	0	8	0	24	0	1
joinCommon(String_String_String_JoinBolt.JoinType)	100%	100%	100%	100%	0	3	0	9	0	1
doProjection(ArrayList_JoinBolt.FieldSelector[])	100%	100%	100%	100%	0	5	0	13	0	1
JoinBolt(JoinBolt.Selector_String_String)	100%	100%	n/a	n/a	0	1	0	6	0	1
select(String)	100%	100%	100%	100%	0	2	0	5	0	1
getJoinField(String_Tuple)	100%	100%	100%	100%	0	2	0	4	0	1
clearHashedInputs()	100%	100%	100%	100%	0	2	0	4	0	1
join(String_String_String)	100%	100%	n/a	n/a	0	1	0	1	0	1
leftJoin(String_String_String)	100%	100%	n/a	n/a	0	1	0	1	0	1
Total	226 of 829	72%	23 of 82	71%	33	74	41	162	16	32

Figura 47

```
@Parameterized.Parameters
public static Collection<Object[]> data() {
    return Arrays.asList(new Object[][] {
        { STREAM.RESERVATIONS, 1, null, 0, JOINTYPE.LEFT, 6 },
        { STREAM.ORDERS, 1, null, 0, JOINTYPE.INNER, 5 },
        { STREAM.ORDERS, 1, null, 0,
JOINTYPE.EMPTY_STRING_JOIN, RuntimeException.class },
        { STREAM.ORDERS, 1, null, 0,
JOINTYPE.NOT_EXISTING_PRIOR, IllegalArgumentException.class },
        { STREAM.NULL, 1, null, 0, JOINTYPE.EMPTY,
NullPointerException.class },
        { STREAM.ORDERS, 1, null, 0,
JOINTYPE.SAME_STREAM_JOIN, IllegalArgumentException.class },
        { STREAM.RESERVATIONS, 1, null, 0,
JOINTYPE.WRONG_LEFT, 6 },
        { STREAM.EMPTY, 0, null, 0, JOINTYPE.WRONG_LEFT,
NullPointerException.class },
    });
}
```

Figura 48

```

<method name="doLeftJoin" desc="(Lorg/apache/storm/bolt/JoinBolt$JoinAccumulator;Ljava/util/Map;
Lorg/apache/storm/bolt/JoinBolt$JoinInfo;Z)Lorg/apache/storm/bolt/JoinBolt$JoinAccumulator;">
<du var="this" def="273" use="286" covered="1"/>
<du var="this" def="273" use="282" covered="0"/>
<du var="buildInput" def="273" use="279" covered="1"/>
<du var="finalJoin" def="273" use="286" covered="1"/>
<du var="finalJoin" def="273" use="282" covered="0"/>
<du var="result" def="274" use="292" covered="1"/>
<du var="result" def="274" use="287" covered="1"/>
<du var="result" def="274" use="283" covered="0"/>
<du var="fieldSelector" def="275" use="277" covered="1"/>
<du var="rec" def="276" use="286" covered="1"/>
<du var="rec" def="276" use="282" covered="0"/>
<du var="probeKey" def="277" use="278" target="279" covered="1"/>
<du var="probeKey" def="277" use="278" target="291" covered="0"/>
<du var="probeKey" def="277" use="279" covered="1"/>
<du var="matchingBuildRecs" def="279" use="280" target="280" covered="0"/>
<du var="matchingBuildRecs" def="279" use="280" target="286" covered="1"/>
<du var="matchingBuildRecs" def="279" use="280" target="281" covered="0"/>
<du var="matchingBuildRecs" def="279" use="280" target="286" covered="0"/>
<du var="matchingBuildRecs" def="279" use="281" covered="0"/>
<counter type="DU" missed="12" covered="13"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 49

```

@Parameterized.Parameters
public static Collection<Object[]> data() {
    return Arrays.asList(new Object[][] {
        { STREAM.RESERVATIONS, 1, STREAM.ORDERS, 0,
JOINTYPE.LEFT, 6 },
        { STREAM.ORDERS, 1, STREAM.EMPTY, 0,
JOINTYPE.INNER, 5 },
        { STREAM.ORDERS, 1, STREAM.NULL, 1,
JOINTYPE.EMPTY_STRING_JOIN, RuntimeException.class },
        { STREAM.ORDERS, 1, STREAM.ORDERS, 1,
JOINTYPE.NOT_EXISTING_PRIOR, IllegalArgumentException.class },
        { STREAM.NULL, 1, STREAM.RESERVATIONS, 1,
JOINTYPE.EMPTY, NullPointerException.class },
        { STREAM.ORDERS, 1, STREAM.ORDERS, 1,
JOINTYPE.SAME_STREAM_JOIN, IllegalArgumentException.class },
        { STREAM.RESERVATIONS, 1, STREAM.ORDERS, 1,
JOINTYPE.WRONG_LEFT, 6 },
        { STREAM.EMPTY, 0, STREAM.RESERVATIONS, 1,
JOINTYPE.WRONC_LEFT, NullPointerException.class },
    });
}

```

Figura 50

```

// left join - core implementation
protected JoinAccumulator doLeftJoin(JoinAccumulator probe, Map<Object, ArrayList<Tuple>> buildInput, JoinInfo joinInfo,
                                     boolean finalJoin) {
    String[] probeKeyName = joinInfo.getOtherField();
    JoinAccumulator result = new JoinAccumulator();
    FieldSelector fieldSelector = new FieldSelector(joinInfo.other.getStreamName(), probeKeyName);
    for (ResultRecord rec : probe.getRecords()) {
        Object probeKey = rec.getField(fieldSelector);
        if (probeKey != null) {
            ArrayList<Tuple> matchingBuildRecs = buildInput.get(probeKey); // ok if its return null
            if (matchingBuildRecs != null && !matchingBuildRecs.isEmpty()) {
                for (Tuple matchingRec : matchingBuildRecs) {
                    ResultRecord mergedRecord = new ResultRecord(rec, matchingRec, finalJoin);
                    result.insert(mergedRecord);
                }
            } else {
                ResultRecord mergedRecord = new ResultRecord(rec, null, finalJoin);
                result.insert(mergedRecord);
            }
        }
    }
    return result;
}

```

Figura 51

```

<method name="doInnerJoin" desc="(Lorg/apache/storm/bolt/JoinBolt$JoinAccumulator;
JoinBolt$JoinInfo;Z)Lorg/apache/storm/bolt/JoinBolt$JoinAccumulator;">
<du var="this" def="252" use="261" covered="1"/>
<du var="buildInput" def="252" use="258" covered="1"/>
<du var="finalJoin" def="252" use="261" covered="1"/>
<du var="result" def="253" use="267" covered="1"/>
<du var="result" def="253" use="262" covered="1"/>
<du var="fieldSelector" def="254" use="256" covered="1"/>
<du var="rec" def="255" use="261" covered="1"/>
<du var="probeKey" def="256" use="257" target="258" covered="1"/>
<du var="probeKey" def="256" use="257" target="266" covered="0"/>
<du var="probeKey" def="256" use="258" covered="1"/>
<du var="matchingBuildRecs" def="258" use="259" target="260" covered="1"/>
<du var="matchingBuildRecs" def="258" use="259" target="266" covered="1"/>
<du var="matchingBuildRecs" def="258" use="260" covered="1"/>
<counter type="DU" missed="1" covered="18"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 52

## JoinBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
declareOutputFields(OutputFieldsDeclarer)	0%	0%	0%	0%	3	3	7	7	1	1
doLeftJoin(JoinBolt.JoinAccumulator, Map<JoinBolt.JoinInfo, boolean>)	67%	40%	40%	40%	4	6	4	16	0	1
getStreamSelector(Tuple)	36%	33%	33%	33%	2	3	2	4	0	1
lookupField(JoinBolt.FieldSelector, Tuple)	75%	66%	66%	66%	3	7	4	13	0	1
doJoin(JoinBolt.JoinAccumulator, HashMap<JoinBolt.JoinInfo, boolean>)	62%	66%	66%	66%	1	3	1	5	0	1
execute(TupleWindow)	79%	75%	75%	75%	1	3	1	9	0	1
JoinBolt(String, String)	0%	n/a	n/a	n/a	1	1	2	2	1	1
withWindow(BaseWindowedBolt.Count, BaseWindowedBolt.Count)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt.Count, BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt.Duration, BaseWindowedBolt.Count)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt.Duration, BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
prepare(Map<TopologyContext, OutputCollector>)	86%	83%	83%	83%	1	4	1	10	0	1
withOutputStream(String)	0%	n/a	n/a	n/a	1	1	2	2	1	1
withWindow(BaseWindowedBolt.Count)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withWindow(BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withTumblingWindow(BaseWindowedBolt.Count)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withTumblingWindow(BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withTimestampField(String)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withTimestampExtractor(TimestampExtractor)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withLateTupleStream(String)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withLag(BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
withWatermarkInterval(BaseWindowedBolt.Duration)	0%	n/a	n/a	n/a	1	1	1	1	1	1
hashJoin(List)	100%	100%	100%	100%	0	8	0	24	0	1
joinCommon(String, String, String, JoinBolt.JoinType)	100%	100%	100%	100%	0	3	0	9	0	1
dolmerJoin(JoinBolt.JoinAccumulator, Map<JoinBolt.JoinInfo, boolean>)	100%	87%	87%	87%	1	5	0	14	0	1
doProjection(ArrayList<JoinBolt.FieldSelector>)	100%	100%	100%	100%	0	5	0	13	0	1
JoinBolt(JoinBolt.Selector, String, String)	100%	n/a	n/a	n/a	0	1	0	6	0	1
select(String)	100%	100%	100%	100%	0	2	0	5	0	1
getJoinField(String, Tuple)	100%	100%	100%	100%	0	2	0	4	0	1
clearHashedInputs()	100%	100%	100%	100%	0	2	0	4	0	1
join(String, String, String)	100%	n/a	n/a	n/a	0	1	0	1	0	1
leftJoin(String, String, String)	100%	n/a	n/a	n/a	0	1	0	1	0	1
Total	204 of 829	75%	20 of 82	75%	31	74	37	162	16	32

Figura 53

```
// inner join - core implementation
protected JoinAccumulator doInnerJoin(JoinAccumulator probe, Map<Object, ArrayList<Tuple>> buildInput, JoinInfo joinInfo,
                                         boolean finalJoin) {
    String[] probeKeyName = joinInfo.getOtherField();
    JoinAccumulator result = new JoinAccumulator();
    FieldSelector fieldSelector = new FieldSelector(joinInfo.other.getStreamName(), probeKeyName);
    for (ResultRecord rec : probe.getRecords()) {
        Object probeKey = rec.getField(fieldSelector);
        if (probeKey != null) {
            ArrayList<Tuple> matchingBuildRecs = buildInput.get(probeKey);
            if (matchingBuildRecs != null) {
                for (Tuple matchingRec : matchingBuildRecs) {
                    ResultRecord mergedRecord = new ResultRecord(rec, matchingRec, finalJoin);
                    result.insert(mergedRecord);
                }
            }
        }
    }
    return result;
}
```

Figura 54

```
// Dispatches to the right join method (inner/left/right/outer) based on the joinInfo.joinType
protected JoinAccumulator doJoin(JoinAccumulator probe, HashMap<Object, ArrayList<Tuple>> buildInput, JoinInfo joinInfo,
                                 boolean finalJoin) {
    final JoinType joinType = joinInfo.getJoinType();
    switch (joinType) {
        case INNER:
            return doInnerJoin(probe, buildInput, joinInfo, finalJoin);
        case LEFT:
            return doLeftJoin(probe, buildInput, joinInfo, finalJoin);
        case RIGHT:
        case OUTER:
        default:
            throw new RuntimeException("Unsupported join type : " + joinType.name());
    }
}
```

Figura 55

```

<method name="select" desc="(Ljava/lang/String;)Lorg/apache/storm/bolt/JoinBolt;">
<du var="this" def="129" use="135" covered="1"/>
<du var="this" def="129" use="133" covered="1"/>
<du var="fieldNames" def="129" use="132" target="133" covered="1"/>
<du var="fieldNames" def="129" use="132" target="135" covered="1"/>
<du var="fieldNames" def="129" use="133" covered="1"/>
<du var="this.outputFields" def="131" use="133" covered="1"/>
<du var="i" def="132" use="133" target="133" covered="1"/>
<du var="i" def="132" use="132" target="135" covered="0"/>
<du var="i" def="132" use="133" covered="1"/>
<du var="i" def="132" use="133" covered="1"/>
<du var="i" def="132" use="132" covered="1"/>
<du var="i" def="132" use="132" target="133" covered="1"/>
<du var="i" def="132" use="132" target="135" covered="1"/>
<du var="i" def="132" use="133" covered="1"/>
<du var="i" def="132" use="133" covered="1"/>
<du var="i" def="132" use="132" covered="1"/>
<counter type="DU" missed="1" covered="15"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 56

```

public JoinBolt select(String commaSeparatedKeys) {
    String[] fieldNames = commaSeparatedKeys.split(",");
    outputFields = new FieldSelector[fieldNames.length];
    for (int i = 0; i < fieldNames.length; i++) {
        outputFields[i] = new FieldSelector(fieldNames[i]);
    }
    return this;
}

```

Figura 57

## Pit Test Coverage Report

### Package Summary

**org.apache.storm.bolt**

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	80% <div style="width: 80%; background-color: #90EE90;"></div>	54% <div style="width: 54%; background-color: #FFCCCC;"></div>	72% <div style="width: 72%; background-color: #90EE90;"></div>

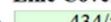
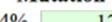
### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">JoinBolt.java</a>	80% <div style="width: 80%; background-color: #90EE90;"></div>	54% <div style="width: 54%; background-color: #FFCCCC;"></div>	72% <div style="width: 72%; background-color: #90EE90;"></div>

Figura 58

# Pit Test Coverage Report

## Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
3	73%  434/595 	44%  121/274 	62%  121/195 

## Breakdown by Package

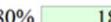
Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
<a href="#">org.apache.storm.bolt</a>	1	80%  182/228 	58%  56/97 	78%  56/72 

Figura 59

#cas	Coverage getTupleType (S-B-DF)	Coverage checkFinishId (S-B-DF)
5	72% - 50% - 52%	29% - 26% - 27%
10	92% - 75% - 70%	29% - 26% - 27%
15	100% - 87% - 83%	31% - 30% - 27%

Tabella

# CASI DI TEST

**Caso 1:** {"test", "part"}  
**Caso 2:** {"", ""}

**Caso 1:** {-1, 0, -1, 0}  
**Caso 2:** {1, 0, 0, 1}  
**Caso 3:** {1, -1, 1, -1}

**Caso 1:** {-1, 1, UnpooledByteBufAllocator.DEFAULT.buffer(1024), true}  
**Caso 2:** {0, 0, UnpooledByteBufAllocator.DEFAULT.buffer(1024), false}  
**Caso 3:** {1, -1, UnpooledByteBufAllocator.DEFAULT.buffer(11\*1024), false}

**Caso 4:** {0, -1, null, false}  
**Caso 5:** {0, 0, UnpooledByteBufAllocator.DEFAULT.buffer(2 \* 1024 + 2), true}  
**Caso 6:** {1, 1, UnpooledByteBufAllocator.DEFAULT.buffer(10 \* 1024 - 1), false}  
**Caso 7:** {-1, 0, UnpooledByteBufAllocator.DEFAULT.buffer(11 \* 1024), false}

**Caso 1:** {0, 0, 0, 0}  
**Caso 2:** {-1, -1, -1, -1}  
**Caso 3:** {1, 0, 0, 1}

**Caso 1:** {*mrb*, “ ”, *CoordinatedBolt.SourceArgs.all()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“ ”, “streamTest”)}*}

**Caso 2:** {*mrb*, “srcTest”, *CoordinatedBolt.SourceArgs.all()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“srcTest”, “streamTest”)*}

**Caso 3:** {*mrb*, “srcTest”, *CoordinatedBolt.SourceArgs.all()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“srcTest”, “ ”)*}

**Caso 4:** {*mrb*, “srcTest”, *CoordinatedBolt.SourceArgs.all()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“srcTest”,*  
*Constants.COORDINATED\_STREAM\_ID)*}

**Caso 5:** {*mrb*, “ ”, *CoordinatedBolt.SourceArgs.all()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“ ”,*  
*Constants.COORDINATED\_STREAM\_ID)*}

**Caso 6:** {*mrb*, “ ”, *CoordinatedBolt.SourceArgs.single()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“ ”, Constants.COORDINATED\_STREAM\_ID)*}

**Caso 7:** {*mrb*, null, *CoordinatedBolt.SourceArgs.single()*, null}

**Caso 8:** {*mrb*, “srcTest”, *CoordinatedBolt.SourceArgs.single()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“srcTest”, “ ”)*}

**Caso 9:** {*mrb*, “srcTest”, “streamTest”, *CoordinatedBolt.SourceArgs.single()*,  
*CoordinatedBolt.IdStreamSpec.makeDetectSpec(“srcTest”, “streamTest”)*}

**Caso 10:** {*mrb*, null, *CoordinatedBolt.SourceArgs.all()*, null}

**Caso 1:** {“ ”, “streamTest”, -1}

**Caso 2:** {"srcTest", "streamTest", 10}

**Caso 3:** {"srcTest", " ", -1}

**Caso 4:** {"srcTest", Constants.COORDINATED\_STREAM\_ID, 0}

**Caso 5:** {" ", Constants.COORDINATED\_STREAM\_ID, 10}

**Caso 6:** {" ", Constants.COORDINATED\_STREAM\_ID, 10}

**Caso 7:** {null, “streamTest”, 0}

**Caso 8:** {"srcTest", " ", -1}

**Caso 9:** {null, " ", -1}

**Caso 10:** {"srcTest", “streamTest”, null}

**Caso 1:** { " ", "streamTest", objectList1}  
**Caso 2:** {"srcTest", "streamTest", objectList1}  
**Caso 3:** {"srcTest", " ", objectList1}  
**Caso 4:** {"srcTest", Constants.COORDINATED\_STREAM\_ID, objectList1}  
**Caso 5:** {" ", Constants.COORDINATED\_STREAM\_ID, objectList1}

**Caso 6:** {" ", Constants.COORDINATED\_STREAM\_ID, objectList2}  
**Caso 7:** {null, "streamTest", objectList2}  
**Caso 8:** {"srcTest", " ", objectList2}  
**Caso 9:** {null, " ", objectList2}  
**Caso 10:** {"srcTest", "streamTest", objectList2}

**Caso 11:** {"srcTest", "streamTest", objectList1}  
**Caso 12:** {"srcTest", "streamTest", objectList1}  
**Caso 13:** {"srcTest", "streamTest", objectList2}  
**Caso 14:** {"srcTest", Constants.COORDINATED\_STREAM\_ID, objectList2}  
**Caso 15:** {"srcTest", " ", objectList2}

**Caso 1:** {JoinBolt.Selector.STREAM, "", "", JoinBolt.class}  
**Caso 2:** {null, "streamName", "keyField", JoinBolt.class},  
**Caso 3:** {JoinBolt.Selector.STREAM, null, null, NullPointerException.class}  
**Caso 4:** {JoinBolt.Selector.SOURCE, "streamName", "keyField", JoinBolt.class}  
**Caso 5:** {JoinBolt.Selector.STREAM, "streamName", "keyField", JoinBolt.class}

**Caso 1:** {"field1,field2", transformStreamData()}  
**Caso 2:** {"field1", transformStreamDataFirstOnly()}  
**Caso 3:** {"", nullStream()}  
**Caso 4:** {null, NullPointerException.class}  
**Caso 5:** {"notExistingField!", nullStream()}  
**Caso 6:** {"field1,field2,field3", transformStreamDataPlusNull()}