

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.

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 ad 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 sottopackage *storage.ldb* del package Bookie e ByteBufList nel package Util, tutte all'interno del modulo *bookkeeper-server*.

Inizialmente era stata testata la classe FileInfo del package Bookie al posto di ReadCache e WriteCache, scelta perché è una classe presente dalla prima versione del progetto e che ha subito molte modifiche nel corso del tempo, ma è stata eliminata poco prima del completamento del progetto per via di problemi che impedivano la corretta analisi della mutation coverage, quindi ho scelto di riportare, sommariamente, questa classe e l'analisi del testing svolto su di essa, limitandomi a riferire il report di JaCoCo.

Anche la scelta di ByteBufList è motivata da ragioni significative, perché nonostante la classe sia più recente, ha anch'essa subito numerosi commit, e questa peculiarità l'ha resa un buon candidato per la presenza di bug, giustificando la sua inclusione nei casi di test.

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

2.3. FileInfo

Questa classe è una componente che svolge un ruolo cruciale nell'organizzazione dei ledger. Un ledger in Bookkeeper è una sequenza di voci o entry ordinate, e ognuna rappresenta un dato o un'informazione specifica. La classe FileInfo è responsabile di mantenere le informazioni di gestione associate a un singolo ledger, mappando gli *entryId* con la loro posizione.

Il file degli indici del ledger si struttura con un header composto da 4 byte che rappresentano la fingerprint del file (sequenza "BKLE"), seguiti dalla lunghezza della master key in byte, la stessa master key in forma di byte, e infine un bitmap di stato racchiuso in ulteriori 4 byte.

La classe di test contiene una fase preliminare per il setup dell'ambiente, in cui viene creata una cartella temporanea che contiene un file vuoto che verrà usato nelle fasi successive.

Le tre funzioni principali per il test sono:

- *testBasic();*
- *testWriteAndRead();*
- *testMoveToNewLocation();*

2.3.1. Individuazione dominio di input e partizionamento dominio

- *testBasic()*: la funzione contiene i test per il controllo dell'assegnamento della masterkey e del file creato nella fase di setup dell'ambiente. Inoltre viene effettuato un assegnamento esplicito del *Lac* (last add confirmed) e viene controllata la correttezza. Il *Lac* assieme alla entry accelera il recupero del ledger nel caso in cui il writer si blocchi.

I metodi testati sono:

- *public FileInfo(File If, byte[] masterKey, int fileInfoVersionToWrite)*: è il costruttore del SUT. Analizzando i parametri di input, il partizionamento del dominio è:

```
If = {null, valid: qualsiasi file creato ed esistente}
masterKey = {null, valid: qualsiasi stringa (anche vuota)}
fileInfoVersionToWrite = {invalid: qualsiasi intero, valid: (V0, V1)}
```

Per *If* viene escluso il valore null, poiché impedisce l'istanziazione della classe stessa; mentre per l'ultimo parametro i valori valid sono valori consigliati, ma viene accettato un qualunque intero come input (anche negativo).

I valori dei vari parametri cambiano la coverage degli altri metodi presenti nella classe. Per il costruttore il valore di statement coverage e branch coverage rimane del 100% per qualunque scelta del parametro in questione ([figura 1](#)).

- *public void setExplicitLac(ByteBuf lac)*: serve a inserire un nuovo valore per *Lac*. Analizzando i parametri di input, il partizionamento del dominio è:

```
lac = {null, valid, not valid}
```

Per il caso valid si parla di un qualunque ByteBuf composto da 8 byte o multiplo di 8 byte, inizialmente solo allocati, che hanno bisogno di essere scritti poiché vale la relazione: *readerIndex + size_buffer < writerIndex*. Se questa non è rispettata solleva l'eccezione *IndexOutOfBoundsException*.

Per l'istanza not valid si intendono ByteBuf di altre size.

Per questo metodo, inizialmente, la statement coverage era del 90% e la branch coverage del 50%.

Migliorando il test, cioè aggiungendo il livello debug del logger, si raggiunge il 100% in statement e branch coverage ([figura 1](#) e [figura 2](#)).

Per quanto riguarda il mutation test, i casi di test coprono i cambiamenti effettuati dal framework e non sono necessarie modifiche ([figura 19](#)) ma, come detto nella presentazione, sono stati riscontrati dei problemi nel mutation testing, quindi da questo punto non verrà più commentata la mutation coverage per la classe *FileInfo*.

- *testWriteAndRead()*: contiene i test per la scrittura e la lettura del file generato in fase di setup dell'ambiente.

Inizialmente viene controllato che il numero di byte scritti sia uguale alla grandezza del file, senza considerare l'header. Poi si verifica che l'operazione di write sia andata a buon fine e, in caso

affermativo, viene effettuata la read e il confronto tra ciò che viene letto e ciò che è stato scritto in precedenza, che deve risultare uguale.

I metodi testati sono:

- `public synchronized long write(ByteBuffer[] buffs, long position)`: ritorna il numero di byte scritti nel file *If*.

Analizzando i parametri di input, il partizionamento del dominio è:

$$\begin{aligned} \text{buffs} &= \{\text{invalid: null, valid: qualsiasi stringa (anche vuota)}\} \\ \text{position} &= \{>0, =0, <\text{lf.size()}\} \end{aligned}$$

Dove *buffs* rappresenta l'array degli oggetti da scrivere, e *position* specifica la posizione all'interno del *FileChannel* in cui si iniziano a scrivere i dati.

Il valore di statement coverage è del 92% mentre quello di branch coverage è dell'83% ([figura 1](#)), non migliorabile poiché un branch mancante è quello relativo al sollevamento di un'eccezione legata a un'eventuale scrittura non effettuata o buffer vuoto, un altro riguarda un'eccezione legata a una scrittura non effettuata o buffer vuoto, e un altro relativo a un controllo della variabile *newSize*, non accessibile.

Nel primo caso c'è un controllo che impedisce la chiamata dell'operazione di write se il buffer di input è vuoto ([figura 3](#)).

- `public int read(ByteBuffer bb, long position, boolean bestEffort)`: ritorna il numero di byte letti.

Analizzando i parametri di input, il partizionamento del dominio è:

$$\begin{aligned} \text{bb} &= \{\text{valid}\} \\ \text{position} &= \{0 < \text{valore} < \text{size del file}\} \\ \text{bestEffort} &= \{\text{false, true}\} \end{aligned}$$

Il parametro *bb* assume come valore solo istanze valide di *ByteBuffer*, in quanto serve a immagazzinare tutti i byte letti; *position* indica da quale distanza in byte iniziare a leggere; *bestEffort* indica la possibilità di terminare il metodo arrivando all'EOF senza sollevare eccezione (true), altrimenti solleva una *ShortReadException* quando la read raggiunge l'EOF.

Questo metodo chiama `readAbsolute(bb, position + START_OF_DATA, bestEffort)` che aggiorna la posizione di partenza della lettura saltando i byte di header.

La statement coverage iniziale di `readAbsolute` è del 64% e la branch coverage è del 50%. Cambiando i parametri di *bestEffort*, aggiungendo il valore true ai parametri del test, e di *bb*, istanziando un oggetto *ByteBuffer* vuoto e allocando solo memoria per far leggere 0 byte al metodo *fc.read*, si raggiunge il massimo valore di coverage possibile, con una statement coverage al 94% e branch coverage all'87% per il metodo `readAbsolute`, mentre la coverage è del 100% per il metodo `read` ([figura 1](#)).

Non si può raggiungere il 100% per entrambe le metriche perché un branch è irraggiungibile, in quanto il metodo `checkopen` gestisce la variabile *fc* che si occupa della creazione del canale con il file da leggere, che pone la condizione nel branch in questione ([figura 4](#)).

- `testMoveToNewLocation()`: sposta il file creato in fase di setup in un nuovo path, cancellando il file precedente.

Il metodo testato è:

- `public synchronized void moveToNewLocation(File newFile, long size)`.

Analizzando i parametri di input, il partizionamento del dominio è:

$$\begin{aligned} \text{newFile} &= \{\text{null, valid}\} \\ \text{size} &= \{<0, =0, >0\} \end{aligned}$$

Il primo parametro nel caso null solleva una *NullPointerException* e il test fallisce.

Il parametro *size* indica la dimensione in byte del contenuto del file da copiare in *newFile*.

La statement coverage è al 70% e la branch coverage al 68% ([figura 1](#)).

I branch non coperti sono irraggiungibili e servono a rivelare i seguenti errori: la creazione di un file temporaneo che usa il metodo per copiare il contenuto del file *If* che non va a buon fine; trasferimento del contenuto del file non avvenuto; cancellazione del file *If* non avvenuta; ridefinizione del file temporaneo non avvenuta.

L'impossibilità di raggiungere questi branch si verifica poiché le condizioni definite all'interno dei rami non sono influenzate né dai parametri in ingresso al metodo né dai parametri in input al costruttore della fase di test ([figura 5](#)).

2.4. 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 sviluppata è composta innanzitutto da una fase di setup, nella quale si crea una cartella temporanea in cui verrà generato un file vuoto da usare nelle fasi successive, e da due funzioni:

- `testGetBytesAndArrays();`
- `testEncoderWithWrite();`.

2.4.1. Individuazione dominio di input e partizionamento dominio

- `testGetBytesAndArrays()`: verifica la creazione di un'istanza ByteBufList contenente due buffer composti da due oggetti di tipo String. Successivamente verifica anche l'inserimento di un oggetto di tipo String all'inizio della lista e che il numero di byte della lista completa sia uguale al numero di byte corrispondente alla somma delle stringhe inserite.

I metodi testati sono:

- `public static ByteBufList get(ByteBuf b1, ByteBuf b2)`: l'oggetto ByteBuf viene allocato tramite `public static ByteBuf wrappedBuffer(@NotNull byte[] array)`. L'oggetto array richiesto in input può assumere come valori delle stringhe qualsiasi, anche una stringa vuota.
- `public void prepend(ByteBuf buf)`: aggiunge un oggetto ByteBuf all'inizio della lista.

Per entrambi i metodi lo studio della coverage e della mutazione è banale.

- `testEncoderWithWrite()`: scrive il contenuto di un buffer in un file e verifica che tale scrittura sia stata eseguita leggendo il file.

Il metodo testato è:

- `public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise)`.
Questo metodo è contenuto nella classe interna *Encoder*.
Il parametro *ctx* è definito tramite un'istanza della classe *ChannelHandlerContext* creata tramite l'utilizzo di Mockito, e gli è associato il comportamento di scrittura sul file creato nella fase di setup del test; il parametro *msg* rappresenta l'oggetto messaggio da scrivere e può assumere qualsiasi valore, anche una stringa vuota; il parametro *promise*, un oggetto canale asincrono per la trasmissione del messaggio, viene impostato come nullo nel test poiché non è rilevante per la scrittura su file.

Inizialmente l'analisi della coverage rivelava una statement coverage del 13%, una branch coverage del 12% ([figura 15](#), [figura 16](#)) e una data flow coverage del 12% (4 su 34).

Per migliorarle, sono state introdotte le seguenti modifiche: l'aggiunta di un comportamento all'oggetto *ctx*, per cui quando viene chiamato *ctx.alloc()* viene

istanziato l'oggetto ByteBufAllocator richiesto dal metodo *directBuffer(int, int)* presente nel metodo di test; l'aggiunta di un parametro *Encoder* per coprire il caso in cui *prependSize* sia true; l'aggiunta di casi di test in cui l'oggetto *msg* è di tipo ByteBufList con dimensione 1 e dimensione 2.

Con queste modifiche si raggiungono statement coverage e branch coverage del 100% ([figura 17](#), [figura 18](#)), e data flow coverage del 94% ([figura 45](#)).

Per quest'ultima il massimo non è stato raggiunto a causa di una variabile locale definita all'interno di un ciclo for irraggiungibile.

Per quanto riguarda il mutation testing, i casi di test non coprono i cambiamenti effettuati da Pit, ma aggiungendo un controllo sul numero di byte letti aumenta la mutation coverage ([figura 20](#)). Le altre mutazioni non coperte non sono raggiungibili.

2.5. 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.

2.5.1. Individuazione dominio di input e partizionamento dominio

Per testare questa 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 del ledger e della entry da inserire nella cache rispetto a quelli utilizzati nel metodo *get()*.

Il dominio viene partizionato in questo modo:

```
ledgerId = {-1, 0, 1}  
entryId = {-1, 0, 1}
```

E i casi di test sviluppati sono stati i seguenti:

ledgerIdGet	entryIdGet	ledgerIdPut	entryIdPut
Caso 1: { -1,	0,	-1,	0 }
Caso 2: { 1,	0,	0,	1 }
Caso 3: { 1,	-1,	1,	-1 }

Dove il terzo è l'unico che non solleva eccezioni, riscontrando un successo del metodo *get()*.

Per quanto riguarda l'analisi della coverage, si può notare che vengono raggiunte statement e branch coverage del 100% per il metodo preso in considerazione ([figura 6](#), [figura 7](#)), e una data flow coverage del 97% ([figura 46](#)).

La mutation coverage della classe *ReadCache* è del 18% ([figura 56](#)) e viene mostrato il report di Pit per il metodo testato *get*.

2.6. WriteCache

Questa classe implementa una cache di scrittura, che alloca la dimensione richiesta della memoria diretta e la suddivide in più segmenti. Le entries 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)*;
- *public ByteBuf get(long ledgerId, long entryId)*.

2.6.1. Individuazione dominio di input e partizionamento dominio

- `public boolean put(long ledgerId, long entryId, ByteBuf entry)`: copia il contenuto del buffer entry nella cache di scrittura, associando l'entryId specifico a un ledger identificato da ledgerId. Questo metodo restituisce true se la scrittura ha successo e false in caso di fallimento.

Si ricorda che il ledger è una sequenza di entries, dove ciascuna entry è a sua volta una sequenza di byte. Queste entries vengono scritte sequenzialmente e possono essere scritte al massimo una volta su un ledger.

I parametri di input `ledgerId` e `entryId` sono di tipo `long`, e dato che non sono specificate ulteriori informazioni sui range accettabili per questi valori, si possono definire due partizioni per ciascun parametro:

```
ledgerId, entryId = {>=0, <0}
entry = {null, new ByteBuf()}
```

Poiché il successo o il fallimento del metodo dipendono dalla quantità di byte da scrivere rispetto alla memoria disponibile nella cache, è importante suddividere i casi in cui la quantità di byte da scrivere è inferiore o superiore alla capacità della cache stessa. Questo comporterà rispettivamente un successo o un fallimento della scrittura.

Il dominio viene partizionato in questo modo:

```
ledgerId = {-1, 0, 1}
entryId = {-1, 0, 1}
```

```
entry = {null, entry size <= available memory in cache, entry size > available memory in cache}
```

E i casi di test sviluppati sono stati i seguenti:

ledgerId	entryId	entry
Caso 1: { 0,	-1,	null }
Caso 2: { -1,	1,	entry with size <= available memory in cache }
Caso 3: { 0,	0 ,	entry with size > available memory in cache }
Caso 4: { 0,	0 ,	entry with size <= available memory in cache }

Dove i primi due casi restituiscono un fallimento del metodo `put()`, poiché sollevano rispettivamente una `NullPointerException` dovuta a `entryId` negativo e una `IllegalArgumentException` dovuta a `ledgerId` negativo.

Il terzo caso porta a un fallimento della scrittura dovuto al fatto che si sta cercando di scrivere nella cache una quantità di memoria superiore a quella disponibile.

L'ultimo caso è l'unico che conduce a un successo nell'operazione di scrittura.

Per quanto riguarda l'analisi della coverage, inizialmente si ha una statement coverage del 96% e una branch coverage del 62% ([figura 9](#), [figura 10](#)).

In seguito ai cambiamenti nei test, si raggiunge una statement coverage del 97% e una branch coverage del 75% ([figura 11](#), [figura 12](#)) e una data flow coverage dell'87.5% ([figura 47](#)).

- `public ByteBuf get(long ledgerId, long entryId)`: legge il contenuto associato a una entry con un identificatore specifico all'interno del ledger avente anch'esso un identificatore. Se i parametri forniti identificano una entry valida, il metodo restituisce un oggetto `ByteBuf` con il contenuto della entry. Altrimenti, se l'entry non è presente in cache o la coppia di `ledgerId` e `entryId` non è valida, il metodo restituirà null.

Durante la fase di setup è stata inizializzata una `WriteCache` in cui sono state scritte delle entry utilizzando il metodo `put`.

Per condurre il test è stato separato l'uso di `ledgerId` e `entryId` da inserire nella cache attraverso il metodo `put` da quelli utilizzato nel metodo `get`.

Il dominio viene partizionato in questo modo:

```
ledgerId = {-1, 0, 1}
entryId = {-1, 0, 1}
```

E i casi di test sviluppati sono stati i seguenti:

```

    ledgerIdGet entryIdGet ledgerIdPut entryIdPut
Caso 1: { -1,          0,          -1,          0}
Caso 2: { 1,           0,           0,           1}
Caso 3: { 1,           -1,          1,          -1}

```

L'ultimo dei tre casi è l'unico che non solleva eccezioni e porta a un successo del metodo `get`.

Per quanto riguarda l'analisi della coverage, si raggiungono statement coverage, branch coverage e data flow coverage del 100% ([figura 11](#), [figura 47](#)).

La mutation coverage della classe `WriteCache` è del 32% ([figura 56](#)) e viene mostrato il report di Pit per i metodi testati [put](#) e [get](#).

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 `WindowedBoltExecutor` e `CoordinatedBolt`, appartenenti rispettivamente ai package `topology` e `coordination` all'interno del modulo `storm-client`.

La selezione di `WindowedBoltExecutor` è giustificata dalla sua caratteristica di avere un valore di churn costantemente elevato nel corso del tempo, indicando delle possibili e contigue aggiunte di nuove funzionalità alla classe.

Dall'altra parte, `CoordinatedBolt` è presente dalla prima versione del progetto e viene sottoposto a numerosi commit nel corso del tempo. Questo fattore motiva la scelta poiché, a causa delle continue modifiche nel suo sviluppo, la classe potenzialmente presenta bug o problemi.

L'analisi dettagliata di queste due classi può fornire informazioni utili sulla stabilità e sull'evoluzione del sistema.

3.3. WindowedBoltExecutor

Questa classe è responsabile dell'esecuzione di un componente di tipo Bolt all'interno di una topologia. In particolare, si occupa di gestire il "windowing" delle tuple ricevute, il quale è un processo basato su una finestra temporale di durata specifica, espressa in un numero di secondi definito.

La classe di test è composta da un solo metodo:

- `testExecuteWithTs()`.

3.3.1. Individuazione dominio di input e partizionamento dominio

`testExecuteWithTs()`: genera un oggetto fittizio di tipo `WindowedBolt`, che funge da input per la classe `WindowedBoltExecutor`. Dopo una fase di configurazione appropriata, il test verifica se l'oggetto fittizio riceve correttamente dei valori numerici tramite l'esecuzione della classe in fase di test. In questo modo, si garantisce che la classe `WindowedBoltExecutor` svolga in modo accurato e coerente la sua funzione di windowing delle tuple in base al periodo temporale specificato.

I metodi testati sono:

- `private WindowManager<Tuple> initWindowManager(WindowLifecycleListener<Tuple> lifecycleListener, Map<String, Object> topoConf, TopologyContext context, Collection<Event<Tuple>> queue, boolean stateful)`: inizializza la finestra; il metodo è controllabile tramite il metodo `public void prepare(Map<String, Object> topoConf,`

TopologyContext context, OutputCollector collector) e serve a inizializzare un oggetto *WindowManager*.

Per quanto riguarda la category partition dei parametri, *topoConf* assume valori coppie chiave-valore che fanno parte della classe *Config*; *context* viene istanziato tramite l'utilizzo del framework Mockito e viene definito il comportamento per il metodo *getThisSources()*, con la funzione di dichiarare da quale input prende i dati ricevuti dall'oggetto Bolt; il parametro *collector* viene istanziato tramite l'utilizzo di Mockito.

Dopo aver aggiunto configurazioni valide per il parametro *topoConf*, un nuovo comportamento al parametro *context* (*context.getThisStream()*) e un controllo sul sollevamento dell'eccezione *IllegalArgumentException*, la statement coverage aumenta al 96%, la branch coverage all'86% ([figura 21](#), [figura 22](#), [figura 23](#)) e la data flow coverage al 91.5% ([figura 48](#)).

Nelle figure [24](#) e [25](#) viene mostrata la mutation coverage dopo aver apportato queste modifiche.

- *public void execute(Tuple input)*: prende in input una tupla, la elabora e la inoltra nella topologia che è stata creata precedentemente nella fase di prepare. Rappresenta una parte cruciale dell'elaborazione all'interno della topologia.

Analizzando i parametri di input, il partizionamento del dominio è:

```
input = {valid, null}
```

Se a input viene dato il valore null, questo solleverà un'eccezione di tipo *NullPointerException*, quindi l'opzione viene esclusa.

L'analisi della coverage rivela che la statement coverage è del 40%, la branch coverage è del 33% e la data flow coverage è del 34%.

Un miglioramento significativo si ottiene includendo un caso in cui viene seguita la execute del Bolt fittizio con il parametro *timeStampExtractor* impostato su null, portando la statement coverage al 47%, la branch coverage al 50% ([figura 21](#), [figura 26](#)) e la data flow coverage al 46% ([figura 49](#)).

La mutation coverage non può essere migliorata ulteriormente poiché alcune parti del codice rimangono inaccessibili ([figura 27](#)).

- *protected void validate(Map<String, Object> topoConf, Count windowLengthCount, Duration windowLengthDuration, Count slidingIntervalCount, Duration slidingIntervalDuration)*: convalida e garantisce la correttezza dei dati relativi al *WindowManager*.

Questo metodo viene chiamato da *initWindowManager* con i suoi parametri.

Il parametro *topoConf* nell'operazione di validate è lo stesso che prende in input il metodo *initWindowManager*, il parametro *windowLengthCount* è il valore della lunghezza della finestra ed è inizializzato in *topoConf*, *windowLengthDuration* è il valore della durata temporale della finestra e viene inizializzato in *topoConf*, *slidingIntervalDuration* indica la durata temporale di ogni divisione e viene inizializzato in *topoConf*.

Riguardo l'analisi della coverage, inizialmente la statement coverage è del 43%, la branch coverage del 35% e la data flow coverage del 75%.

Si ottiene un aumento di queste percentuali mediante l'aggiunta di casi di test che variano i parametri, portando la statement coverage al 78%, la branch coverage all'80% ([figura 21](#), [figura 37](#)) e lasciando invariata la data flow coverage ([figura 50](#)).

La mutation coverage del metodo prima e dopo queste aggiunte sopra citate rimane la stessa ([figura 38](#)).

Per la classe *WindowedBoltExecutor*, la mutation coverage totale è del 55% ([figura 58](#)).

3.4. 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 dai metodi:

- `testPrepareAndExecute();`
- `testPrepareAndExecuteDiffConstructor();`

3.4.1. Individuazione dominio di input e partizionamento dominio

- `testPrepareAndExecute()`: verifica il corretto inserimento dei parametri `srcComponent` e `streamId` nella classe di test.
- `testPrepareAndExecuteDiffConstructor()`: istanzia un Bolt delegato fittizio (`MockRichBoltTimeout`), il quale riceve le tuple inviate dalla classe di test.

I metodi testati sono:

- `public CoordinatedBolt(IRichBolt delegate, String sourceComponent, SourceArgs sourceArgs, IdStreamSpec idStreamSpec)`: uno dei tre costruttori del SUT.

Analizzando i parametri di input, `delegate` prende un oggetto Bolt e deve avere un'istanza valida, senza poter assumere valore null, altrimenti solleverebbe l'eccezione `NullPointerException`; `sourceComponent` serve ad associare un nome alla sorgente dati e può assumere valori `{valid = qualsiasi stringa, invalid = null}`; `sourceArgs` può assumere valori come `{valid = SourceArgs.single(), sourceArgs.all(), null}` e sono valori presenti nella sottoclasse `sourceArgs`, contenuta nella classe di test; `idStreamSpec` può assumere valori `{valid = CoordinatedBolt.idStreamSpec.makeDetectSpec(String sourceComponent, String streamId), null}` dove le stringhe possono assumere come valore una qualsiasi stringa, purché siano uguali a quelle inserite tramite `TopologyContext`, e il parametro si riferisce a una classe interna alla classe di test, con il compito di creare un oggetto `GlobalStreamId` che servirà poi a creare un riferimento allo stream di output del Bolt, prima di dichiarare la topologia.

- `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`.

Per quanto riguarda lo studio dei parametri di input, `config` è un oggetto `HashMap<>` vuoto, in quanto ai fini del test non c'era bisogno di aggiungere particolari parametri di configurazione; `context` è istanziato tramite l'utilizzo del framework Mockito, attraverso il quale vengono definiti dei comportamenti volti ad aumentare la coverage dei metodi contenuti nel SUT.

Inizialmente la statement coverage è del 40%, la branch coverage del 25% ([figura 28](#), [figura 29](#)) e la data flow coverage del 36%.

Successivamente alla variazione di alcuni parametri di test come `sourceArgs` è stato possibile raggiungere una statement coverage dell'81%, una branch coverage del 75% ([figura 30](#), [figura 31](#)) e una data flow coverage del 76% ([figura 51](#)).

Inoltre è stata riportata la mutation coverage prima ([figura 39](#)) e dopo ([figura 40](#)) la variazione dei parametri.

- `public void execute(Tuple tuple)`: questo metodo prende in input una tupla, la elabora e la emette nella topologia creata nella fase di `prepare`.

Nei test, quando la classe del Bolt fittizio è `MockRichBolt`, questo metodo si occupa di inviare le tuple all'oggetto `MockRichBolt` e verifica la ricezione delle stesse. Tuttavia, quando la classe del Bolt fittizio è `MockRichBoltTimeout`, il metodo privato `boolean checkFinishId(Tuple tup, TupleType type)` viene chiamato per instradare le tuple al collector, tenendo traccia di esse tramite il loro identificatore, che è il primo valore contenuto nella tupla.

L'unico parametro di input `tuple` viene istanziato utilizzando la classe `TupleImpl` di Storm e il suo costruttore `public TupleImpl(GeneralTopologyContext context, List<Object> values, String srcComponent, int taskId, String streamId)`. L'oggetto `context` viene istanziato con campi `fields` che identificano i dati nella tupla. Il parametro `values` contiene i dati effettivi da inserire nella tupla, mentre i parametri `srcComponent` e `streamId` devono

essere identici sia per il costruttore del SUT sia per il *context*. Infine, il parametro *taskId* può assumere qualsiasi valore intero che sia maggiore o uguale a zero.

Inizialmente la statement coverage è del 54%, la branch coverage del 50% ([figura 28](#)) e la data flow coverage del 38%. Grazie alle variazioni nei parametri di test come *idStreamSpec* e *sourceArgs* e all'inclusione del *MockRichBoltTimeout*, i valori sono aumentati al 100% per la statement coverage, all'87% per la branch coverage ([figura 30](#), [figura 32](#)) e all'88% per la data flow coverage ([figura 52](#)).

Inoltre è stata riportata la mutation coverage prima ([figura 41](#)) e dopo ([figura 42](#)) la variazione dei parametri.

- *private TupleType getTupleType(Tuple tuple)*: questo metodo opera sulla tipologia di tuple utilizzate all'interno del SUT. Questa è rappresentata da un'enumerazione denominata *TupleType* e comprende tre valori: *Regular*, *Id* e *Coord*.

Quando il costruttore base del SUT viene utilizzato, richiedendo solo il Bolt delegato come input, il *TupleType* assegnato alla tupla passata come input nel metodo *execute* è sempre *Regular*. Le altre due tipologie vengono assegnate in base ai valori specificati nei parametri *sourceArgs* e *idStreamSpec* nel secondo costruttore, descritto in precedenza.

In partenza si ha una statement coverage del 36%, branch coverage del 25% ([figura 28](#)) e data flow coverage del 22%.

In seguito a variazioni nei parametri di test come *sourceArgs* e *idStreamSpec*, e modificando alcuni parametri relativi alla classe *Tuple*, queste percentuali sono aumentate, arrivando a raggiungere una statement coverage del 100%, una branch coverage dell'87% ([figura 30](#), [figura 33](#)) e una data flow coverage dell'83% ([figura 53](#)). Per quanto riguarda la mutation coverage, viene mostrata prima ([figura 43](#)) e dopo ([figura 44](#)) le modifiche introdotte nei test.

Va notato che tutti e tre i metodi del SUT non mostrano un miglioramento significativo nella mutation coverage, poiché coinvolgono variabili e oggetti privati del SUT che non sono accessibili tramite i test, risultando quindi irraggiungibili per il controllo delle mutazioni.

Per la classe *CoordinatedBolt*, la mutation coverage totale è del 14% ([figura 57](#)).

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_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)

FileInfo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
readHeader()	42%	42%	40%	40%	10	12	18	41	0	1
checkOpen(boolean, boolean)	40%	40%	38%	38%	8	10	10	25	0	1
moveToNewLocation(File, long)	70%	70%	68%	68%	6	12	8	32	0	1
setFenced()	0%	0%	0%	0%	4	4	15	15	1	1
waitForLastAddConfirmedUpdate(long, Watcher)	0%	0%	0%	0%	5	5	6	6	1	1
checkParents(File)	0%	0%	0%	0%	3	3	6	6	1	1
isFenced()	0%	0%	0%	0%	2	2	2	2	1	1
setLastAddConfirmed(long)	85%	85%	62%	62%	3	5	1	12	0	1
write(ByteBuffer[], long)	92%	92%	83%	83%	1	4	1	15	0	1
getExplicitLac()	88%	88%	75%	75%	1	3	2	12	0	1
cancelWaitForLastAddConfirmedUpdate(Watcher)	0%	0%	n/a	n/a	1	1	2	2	1	1
readAbsolute(ByteBuffer, long, boolean)	94%	94%	87%	87%	1	5	1	18	0	1
close(boolean)	93%	93%	50%	50%	4	5	1	16	0	1
getLastAddConfirmed()	0%	0%	n/a	n/a	1	1	1	1	1	1
isClosed()	0%	0%	n/a	n/a	1	1	1	1	1	1
getSizeSinceLastWrite()	0%	0%	n/a	n/a	1	1	1	1	1	1
size()	87%	87%	50%	50%	1	2	1	5	0	1
writeHeader()	100%	100%	100%	100%	0	3	0	17	0	1
setExplicitLac(ByteBuf)	100%	100%	100%	100%	0	3	0	14	0	1
FileInfo(File, byte[], int)	100%	100%	n/a	n/a	0	1	0	10	0	1
flushHeader()	100%	100%	100%	100%	0	2	0	5	0	1
static {...}	100%	100%	n/a	n/a	0	1	0	2	0	1
read(ByteBuffer, long, boolean)	100%	100%	n/a	n/a	0	1	0	1	0	1
delete()	100%	100%	n/a	n/a	0	1	0	2	0	1
getMasterKey()	100%	100%	n/a	n/a	0	1	0	2	0	1
checkOpen(boolean)	100%	100%	n/a	n/a	0	1	0	2	0	1
isSameFile(File)	100%	100%	n/a	n/a	0	1	0	1	0	1
getLf()	100%	100%	n/a	n/a	0	1	0	1	0	1
isDeleted()	100%	100%	n/a	n/a	0	1	0	1	0	1
Total	401 of 1.153	65%	62 of 128	51%	53	93	77	268	8	29

Figura 1

```

186.     public void setExplicitLac(ByteBuf lac) {
187.         long explicitLacValue;
188.         synchronized (this) {
189.             ◆ if (explicitLac == null) {
190.                 explicitLac = ByteBuffer.allocate(lac.capacity());
191.             }
192.             lac.readBytes(explicitLac);
193.             explicitLac.rewind();
194.
195.             // skip the ledger id
196.             explicitLac.getLong();
197.             explicitLacValue = explicitLac.getLong();
198.             explicitLac.rewind();
199.             ◆ if (LOG.isDebugEnabled()) {
200.                 LOG.debug("fileInfo:SetLac: {}", explicitLac);
201.             }
202.             needFlushHeader = true;
203.         }
204.         setLastAddConfirmed(explicitLacValue);
205.     }

```

Figura 2

```
476.     public synchronized long write(ByteBuffer[] buffs, long position) throws IOException {
477.         checkOpen(true);
478.         long total = 0;
479.         try {
480.             fc.position(position + START_OF_DATA);
481.             while (buffs[buffs.length - 1].remaining() > 0) {
482.                 long rc = fc.write(buff);
483.                 if (rc <= 0) {
484.                     throw new IOException("Short write");
485.                 }
486.                 total += rc;
487.             }
488.         } finally {
489.             fc.force(true);
490.             long newsize = position + START_OF_DATA + total;
491.             if (newsize > size) {
492.                 size = newsize;
493.             }
494.         }
495.         sizeSinceLastWrite = fc.size();
496.         return total;
497.     }
```

Figura 3

```

394.     public int read(ByteBuffer bb, long position, boolean bestEffort)
395.             throws IOException {
396.         return readAbsolute(bb, position + START_OF_DATA, bestEffort);
397.     }
398.
399.    /**
400.     * Read data from position <i>start</i> to fill the byte buffer <i>bb</i>.
401.     * If <i>bestEffort </i> is provided, it would return when it reaches EOF.
402.     * Otherwise, it would throw {@link org.apache.bookkeeper.bookie.ShortReadException}
403.     * if it reaches EOF.
404.     *
405.     * @param bb
406.     *         byte buffer of data
407.     * @param start
408.     *         start position to read data
409.     * @param bestEffort
410.     *         flag indicates if it is a best-effort read
411.     * @return number of bytes read
412.     * @throws IOException
413.     */
414.    private int readAbsolute(ByteBuffer bb, long start, boolean bestEffort)
415.            throws IOException {
416.        checkOpen(false);
417.        synchronized (this) {
418.            if (fc == null) {
419.                return 0;
420.            }
421.        }
422.        int total = 0;
423.        int rc = 0;
424.        while (bb.remaining() > 0) {
425.            synchronized (this) {
426.                rc = fc.read(bb, start);
427.            }
428.            if (rc <= 0) {
429.                if (bestEffort) {
430.                    return total;
431.                } else {
432.                    throw new ShortReadException("Short read at " + getLf().getPath() + "@" + start);
433.                }
434.            }
435.            total += rc;
436.            // should move read position
437.            start += rc;
438.        }
439.        return total;
440.    }

```

Figura 4

```

504.     public synchronized void moveToNewLocation(File newFile, long size) throws IOException {
505.         checkOpen(false);
506.         // If the channel is null, or same file path, just return.
507.         if (null == fc || isSameFile(newFile)) {
508.             return;
509.         }
510.         if (size > fc.size()) {
511.             size = fc.size();
512.         }
513.         File rlocFile = new File(newFile.getParentFile(), newFile.getName() + IndexPersistenceMgr.RLOC);
514.         if (!rlocFile.exists()) {
515.             checkParents(rlocFile);
516.             if (!rlocFile.createNewFile()) {
517.                 throw new IOException("Creating new cache index file " + rlocFile + " failed");
518.             }
519.         }
520.         // copy contents from old.idx to new.idx.rloc
521.         FileChannel newFc = new RandomAccessFile(rlocFile, "rw").getChannel();
522.         try {
523.             long written = 0;
524.             while (written < size) {
525.                 long count = fc.transferTo(written, size, newFc);
526.                 if (count <= 0) {
527.                     throw new IOException("Copying to new location " + rlocFile + " failed");
528.                 }
529.                 written += count;
530.             }
531.             if (written <= 0 && size > 0) {
532.                 throw new IOException("Copying to new location " + rlocFile + " failed");
533.             }
534.         } finally {
535.             newFc.force(true);
536.             newFc.close();
537.         }
538.         // delete old.idx
539.         fc.close();
540.         if (!delete()) {
541.             LOG.error("Failed to delete the previous index file " + lf);
542.             throw new IOException("Failed to delete the previous index file " + lf);
543.         }
544.
545.         // rename new.idx.rloc to new.idx
546.         if (!rlocFile.renameTo(newFile)) {
547.             LOG.error("Failed to rename " + rlocFile + " to " + newFile);
548.             throw new IOException("Failed to rename " + rlocFile + " to " + newFile);
549.         }
550.         fc = new RandomAccessFile(newFile, mode).getChannel();
551.         lf = newFile;
552.     }

```

Figura 5

ReadCache

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

Figura 6

```

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 7

```

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 8

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)	0%	0%	0%	0%	8	8	32	32	1	1
get(long, long)	0%	0%	0%	0%	2	2	10	10	1	1
lambda\$forEach\$0(long, long, long, long)	0%	0%	0%	0%	2	2	8	8	1	1
getLastEntry(long)	0%	0%	0%	0%	2	2	4	4	1	1
WriteCache(ByteBufAllocator, long, int)	92%	92%	50%	50%	3	4	1	25	0	1
isEmpty()	0%	0%	0%	0%	2	2	1	1	1	1
deleteLedger(long)	0%	0%	n/a	n/a	1	1	2	2	1	1
size()	0%	n/a	n/a	n/a	1	1	1	1	1	1
count()	0%	n/a	n/a	n/a	1	1	1	1	1	1
put(long, long, ByteBuf)	96%	96%	62%	62%	3	5	3	20	0	1
clear()	100%	n/a	0	0	1	0	7	0	1	1
close()	100%	100%	0	100%	0	2	0	3	0	1
alignToPowerOfTwo(long)	100%	n/a	0	1	0	1	0	1	0	1
static {...}	100%	n/a	0	1	0	2	0	2	0	1
align64(int)	100%	n/a	0	1	0	1	0	1	0	1
WriteCache(ByteBufAllocator, long)	100%	n/a	0	1	0	1	0	2	0	1
Total	321 of 617	47%	28 of 38	26%	25	35	63	120	8	16

Figura 9

```

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;
        }

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

```

Figura 10

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)	0%	0%	0%	0%	8	8	32	32	1	1
lambda\$forEach\$0(long, long, long, long)	0%	0%	0%	0%	2	2	8	8	1	1
getLastEntry(long)	0%	0%	0%	0%	2	2	4	4	1	1
isEmpty()	0%	0%	0%	0%	2	2	1	1	1	1
deleteLedger(long)	0%	n/a	1	1	2	2	1	1	1	1
size()	0%	n/a	1	1	1	1	1	1	1	1
count()	0%	n/a	1	1	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)	98%	66%	2	4	0	25	0	0	1	1
put(long, long, ByteBuf)	97%	75%	2	5	2	20	0	0	1	1
get(long, long)	100%	100%	0	2	0	10	0	0	1	1
clear()	100%	n/a	0	1	0	7	0	0	1	1
close()	100%	100%	0	2	0	3	0	0	1	1
alignToPowerOfTwo(long)	100%	n/a	0	1	0	1	0	0	1	1
static {...}	100%	n/a	0	1	0	2	0	0	1	1
align64(int)	100%	n/a	0	1	0	1	0	0	1	1
WriteCache(ByteBufAllocator, long)	100%	n/a	0	1	0	2	0	0	1	1
Total	264 of 617	57%	24 of 38	36%	21	35	51	120	7	16

Figura 11

```

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 12

```

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 13

```

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 14

ByteBufList.Encoder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
• write(ChannelHandlerContext, Object, ChannelPromise)	13%	12%	4	5	11	14	0
• ByteBufList.Encoder(boolean)	100%	n/a	0	1	0	3	0
Total	57 of 72	20%	7 of 8	12%	4	6	11

Figura 15

```

@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 16

ByteBufList.Encoder

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Ctry	Missed	Lines	Missed	Methods
• write(ChannelHandlerContext, Object, ChannelPromise)	100%	100%	0	5	0	14	0	1		
• ByteBufList.Encoder(boolean)	100%	n/a	0	1	0	3	0	1		
Total	0 of 72	100%	0 of 8	100%	0	6	0	17	0	2

Figura 17

```

300.     @Override
301.     public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception {
302.         if (msg instanceof ByteBufList) {
303.             ByteBufList b = (ByteBufList) msg;
304.
305.             try {
306.                 if (prependSize) {
307.                     // Prepend the frame size before writing the buffer list, so that we only have 1 single size
308.                     // header
309.                     ByteBuf sizeBuffer = ctx.alloc().directBuffer(4, 4);
310.                     sizeBuffer.writeInt(b.readableBytes());
311.                     ctx.write(sizeBuffer, ctx.voidPromise());
312.                 }
313.
314.                 // Write each buffer individually on the socket. The retain() here is needed to preserve the fact
315.                 // that ByteBuf are automatically released after a write. If the ByteBufPair ref count is increased
316.                 // and it gets written multiple times, the individual buffers refcount should be reflected as well.
317.                 int buffersCount = b.buffers.size();
318.                 for (int i = 0; i < buffersCount; i++) {
319.                     ByteBuf bx = b.buffers.get(i);
320.                     // Last buffer will carry on the final promise to notify when everything was written on the
321.                     // socket
322.                     ctx.write(bx.retainDuplicate(), i == (buffersCount - 1) ? promise : ctx.voidPromise());
323.                 }
324.             } finally {
325.                 ReferenceCountUtil.safeRelease(b);
326.             }
327.         } else {
328.             ctx.write(msg, promise);
329.         }
330.     }
331. }
332.
333. }

```

Figura 18

```

186     public void setExplicitLac(ByteBuf lac) {
187         long explicitLacValue;
188         synchronized (this) {
189             if (explicitLac == null) {
190                 explicitLac = ByteBuffer.allocate(lac.capacity());
191             }
192             lac.readBytes(explicitLac);
193             explicitLac.rewind();
194
195             // skip the ledger id
196             explicitLac.getLong();
197             explicitLacValue = explicitLac.getLong();
198             explicitLac.rewind();
199             if (LOG.isDebugEnabled()) {
200                 LOG.debug("fileInfo:SetLac: {}", explicitLac);
201             }
202             needFlushHeader = true;
203         }
204         setLastAddConfirmed(explicitLacValue);
205     }

```

Figura 19

```

300     @Override
301     public void write(ChannelHandlerContext ctx, Object msg, ChannelPromise promise) throws Exception {
302         if (msg instanceof ByteBufList) {
303             ByteBufList b = (ByteBufList) msg;
304
305             try {
306                 if (prependSize) {
307                     // Prepend the frame size before writing the buffer list, so that we only have 1 single size
308                     // header
309                     ByteBuf sizeBuffer = ctx.alloc().directBuffer(4, 4);
310                     sizeBuffer.writeInt(b.readableBytes());
311                     ctx.write(sizeBuffer, ctx.voidPromise());
312                 }
313
314                 // Write each buffer individually on the socket. The retain() here is needed to preserve the fact
315                 // that ByteBuf are automatically released after a write. If the ByteBufPair ref count is increased
316                 // and it gets written multiple times, the individual buffers refcount should be reflected as well.
317                 int buffersCount = b.buffers.size();
318                 for (int i = 0; i < buffersCount; i++) {
319                     ByteBuf bx = b.buffers.get(i);
320                     // Last buffer will carry on the final promise to notify when everything was written on the
321                     // socket
322                     ctx.write(bx.retainedDuplicate(), i == (buffersCount - 1) ? promise : ctx.voidPromise());
323                 }
324             } finally {
325                 ReferenceCountUtil.safeRelease(b);
326             }
327         } else {
328             ctx.write(msg, promise);
329         }
330     }
331 }

```

Figura 20

WindowedBoltExecutor

Element	Missed Instructions	Cov	Missed Branches	Cov	Missed	Ctxy	Missed	Lines	Missed	Methods	
execute(Tuple)	██████	47%	██████	50%	2	4	4	11	0	1	
declareOutputFields(OutputFieldsDeclarer)	██████	0%	██████	0%	2	2	5	5	1	1	
ensureDurationLessThanTimeout(int, int)	██████	16%	██████	50%	1	2	1	3	0	1	
ensureCountLessThanMaxPending(int, int)	██████	16%	██████	50%	1	2	1	3	0	1	
validate(Map<BaseWindowedBoltCount, BaseWindowedBoltDuration>, BaseWindowedBoltCount, BaseWindowedBoltDuration)	██████	78%	██████	80%	4	11	3	17	0	1	
boltExecute(Supplier<Supplier<Supplier<Long>>)	██████	0%	n/a	1	1	2	2	1	1	1	
getTopologyTimeoutMillis(Map)	██████	66%	██████	33%	3	4	3	8	0	1	
getComponentConfiguration()	██████	0%	██████	0%	2	2	1	1	1	1	
getTriggerPolicy(BaseWindowedBoltCount, BaseWindowedBoltDuration, WindowManager, EvictionPolicy)	██████	81%	██████	83%	1	4	1	7	0	1	
initWindowManager(WindowLifecycleListener, Map<TopologyContext, Collection<boolean>>)	██████	96%	██████	86%	3	12	1	39	0	1	
getEvictionPolicy(BaseWindowedBoltCount, BaseWindowedBoltDuration)	██████	82%	██████	83%	1	4	1	7	0	1	
getMaxSpoutPending(Map)	██████	57%	██████	50%	1	2	1	4	0	1	
restoreState(Map)	██████	0%	n/a	1	1	2	2	1	1	1	
getState()	██████	0%	n/a	1	1	1	1	1	1	1	
getWindowManager()	██████	0%	n/a	1	1	1	1	1	1	1	
doPrepare(Map<TopologyContext, OutputCollector>, Collection<boolean>)	██████	100%	n/a	0	1	0	11	0	1	1	
getComponentStreams(TopologyContext)	██████	100%	██████	75%	1	3	0	6	0	1	
getWindowStartTs(Long)	██████	100%	██████	75%	1	3	0	4	0	1	
start()	██████	100%	██████	100%	0	2	0	6	0	1	
boltExecute(List<List<List<Long>>)	██████	100%	n/a	0	1	0	2	0	1	1	
cleanup()	██████	100%	██████	50%	1	2	0	5	0	1	
WindowedBoltExecutor(IWindowedBolt)	██████	100%	n/a	0	1	0	4	0	1	1	
prepare(Map<TopologyContext, OutputCollector>)	██████	100%	n/a	0	1	0	2	0	1	1	
isTupleTs()	██████	100%	██████	100%	0	2	0	1	0	1	
newWindowLifecycleListener()	██████	100%	n/a	0	1	0	1	0	1	1	
static { ... }	██████	100%	n/a	0	1	0	1	0	1	1	
Total		183 of 724	74%	26 of 90	71%	28	71	28	154	6	26

Figura 21

```
149.     private WindowManager<Tuple> initWindowManager(WindowLifecycleListener<Tuple> lifecycleListener, Map<String, Object> topoConf,
150.                                                 TopologyContext context, Collection<Event<Tuple>> queue, boolean stateful) {
151.
152.     ◆◆◆ WindowManager<Tuple> manager = stateful
153.           ? new StatefulWindowManager<x>(lifecycleListener, queue)
154.           : new WindowManager<x>(lifecycleListener, queue);
155.
156.     Count windowLengthCount = null;
157.     Duration slidingIntervalDuration = null;
158.     Count slidingIntervalCount = null;
159.     // window length
160.     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_COUNT)) {
161.         windowLengthCount = new Count(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_COUNT)).intValue());
162.     } else if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_DURATION_MS)) {
163.         windowLengthDuration = new Duration(
164.             ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_DURATION_MS)).intValue(),
165.             TimeUnit.MILLISECONDS);
166.     }
167.     // sliding interval
168.     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_COUNT)) {
169.         slidingIntervalCount = new Count(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_COUNT)).intValue());
170.     } else if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_DURATION_MS)) {
171.         slidingIntervalDuration =
172.             new Duration(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_DURATION_MS)).intValue(), TimeUnit.MILLISECONDS);
173.     } else {
174.         // default is a sliding window of count 1
175.         slidingIntervalCount = new Count(1);
176.     }
177.     // tuple ts
178.     ◆◆ if (timestampExtractor != null) {
179.         // late tuple stream
180.         lateTupleStream = (String) topoConf.get(Config.TOPOLOGY_BOLTS_LATE_TUPLE_STREAM);
181.     } else if (lateTupleStream != null) {
182.         if (!context.getThisStreams().contains(lateTupleStream)) {
183.             throw new IllegalArgumentException(
184.                 "Stream for late tuples must be defined with the builder method withLateTupleStream");
185.         }
186.     }
187.     // max lag
188.     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_TUPLE_TIMESTAMP_MAX_LAG_MS)) {
189.         maxLagMs = ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_TUPLE_TIMESTAMP_MAX_LAG_MS)).intValue();
```

Figura 22

```

190.     } else {
191.         maxLagMs = DEFAULT_MAX_LAG_MS;
192.     }
193.     // watermark interval
194.     int watermarkInterval;
195.     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WATERMARK_EVENT_INTERVAL_MS)) {
196.         watermarkInterval = ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WATERMARK_EVENT_INTERVAL_MS)).intValue();
197.     } else {
198.         watermarkInterval = DEFAULT_WATERMARK_EVENT_INTERVAL_MS;
199.     }
200.     waterMarkEventGenerator = new WaterMarkEventGenerator<>(manager, watermarkInterval,
201.                                                               maxLagMs, getComponentStreams(context));
202.     } else {
203.         if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_LATE_TUPLE_STREAM)) {
204.             throw new IllegalArgumentException("Late tuple stream can be defined only when specifying a timestamp field");
205.         }
206.     }
207.     // validate
208.     validate(topoConf, windowLengthCount, windowLengthDuration,
209.              slidingIntervalCount, slidingIntervalDuration);
210.     evictionPolicy = getEvictionPolicy(windowLengthCount, windowLengthDuration);
211.     triggerPolicy = getTriggerPolicy(slidingIntervalCount, slidingIntervalDuration,
212.                                       manager, evictionPolicy);
213.     manager.setEvictionPolicy(evictionPolicy);
214.     manager.setTriggerPolicy(triggerPolicy);
215.     return manager;
216. }

```

Figura 23

```

149     private WindowManager<Tuple> initWindowManager(WindowLifecycleListener<Tuple> lifecycleListener, Map<String, Object> topoConf,
150                                         TopologyContext context, Collection<Event<Tuple>> queue, boolean stateful) {
151
152     ◆◆ WindowManager<Tuple> manager = stateful
153         ? new StatefulWindowManager<>(lifecycleListener, queue)
154         : new WindowManager<>(lifecycleListener, queue);
155
156     Count windowLengthCount = null;
157     Duration slidingIntervalDuration = null;
158     Count slidingIntervalCount = null;
159     // window length
160     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_COUNT)) {
161         windowLengthCount = new Count(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_COUNT)).intValue());
162     } else if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_DURATION_MS)) {
163         windowLengthDuration = new Duration(
164             ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WINDOW_LENGTH_DURATION_MS)).intValue(),
165             TimeUnit.MILLISECONDS);
166     }
167     // sliding interval
168     ◆◆ if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_COUNT)) {
169         slidingIntervalCount = new Count(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_COUNT)).intValue());
170     } else if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_DURATION_MS)) {
171         slidingIntervalDuration =
172             new Duration(((Number) topoConf.get(Config.TOPOLOGY_BOLTS_SLIDING_INTERVAL_DURATION_MS)).intValue(), TimeUnit.MILLISECONDS);
173     } else {
174         // default is a sliding window of count 1
175         slidingIntervalCount = new Count(1);
176     }
177     // tuple ts
178     ◆◆ if (timestampExtractor != null) {
179         // late tuple stream
180         lateTupleStream = (String) topoConf.get(Config.TOPOLOGY_BOLTS_LATE_TUPLE_STREAM);
181     } if (lateTupleStream != null) {
182         if (!context.getThisStreams().contains(lateTupleStream)) {
183             throw new IllegalArgumentException(
184                 "Stream for late tuples must be defined with the builder method withLateTupleStream");
185         }
186     }

```

Figura 24

```

187        // max lag
188    if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_TUPLE_TIMESTAMP_MAX_LAG_MS)) {
189        maxLagMs = ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_TUPLE_TIMESTAMP_MAX_LAG_MS)).intValue();
190    } else {
191        maxLagMs = DEFAULT_MAX_LAG_MS;
192    }
193    // watermark interval
194    int watermarkInterval;
195    if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_WATERMARK_EVENT_INTERVAL_MS)) {
196        watermarkInterval = ((Number) topoConf.get(Config.TOPOLOGY_BOLTS_WATERMARK_EVENT_INTERVAL_MS)).intValue();
197    } else {
198        watermarkInterval = DEFAULT_WATERMARK_EVENT_INTERVAL_MS;
199    }
200    waterMarkEventGenerator = new WaterMarkEventGenerator<>(manager, watermarkInterval,
201                                                               maxLagMs, getComponentStreams(context));
202} else {
203    if (topoConf.containsKey(Config.TOPOLOGY_BOLTS_LATE_TUPLE_STREAM)) {
204        throw new IllegalArgumentException("Late tuple stream can be defined only when specifying a timestamp field");
205    }
206}
207// validate
208 validate(topoConf, windowLengthCount, windowLengthDuration,
209           slidingIntervalCount, slidingIntervalDuration);
210 evictionPolicy = getEvictionPolicy(windowLengthCount, windowLengthDuration);
211 triggerPolicy = getTriggerPolicy(slidingIntervalCount, slidingIntervalDuration,
212                                   manager, evictionPolicy);
213 manager.setEvictionPolicy(evictionPolicy);
214 manager.setTriggerPolicy(triggerPolicy);
215 return manager;
216}

```

Figura 25

```

305.    @Override
306.    public void execute(Tuple input) {
307.        if (isTupleTs()) {
308.            long ts = timestampExtractor.extractTimestamp(input);
309.            if (waterMarkEventGenerator.track(input.getSourceGlobalStreamId(), ts)) {
310.                windowManager.add(input, ts);
311.            } else {
312.                if (lateTupleStream != null) {
313.                    windowedOutputCollector.emit(lateTupleStream, input, new Values(input));
314.                } else {
315.                    LOG.info("Received a late tuple {} with ts {}. This will not be processed.", input, ts);
316.                }
317.                windowedOutputCollector.ack(input);
318.            }
319.        } else {
320.            windowManager.add(input);
321.        }
322.    }

```

Figura 26

```

305    @Override
306    public void execute(Tuple input) {
307    if (isTupleTs()) {
308        long ts = timestampExtractor.extractTimestamp(input);
309        if (waterMarkEventGenerator.track(input.getSourceGlobalStreamId(), ts)) {
310        windowManager.add(input, ts);
311        } else {
312        if (lateTupleStream != null) {
313            windowedOutputCollector.emit(lateTupleStream, input, new Values(input));
314        } else {
315            LOG.info("Received a late tuple {} with ts {}. This will not be processed.", input, ts);
316        }
317        windowedOutputCollector.ack(input);
318        }
319    } else {
320        windowManager.add(input);
321    }
322}

```

Figura 27

CoordinatedBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkFinishId(Tuple, CoordinatedBolt.TupleType)	0%	26%	0%	22	22	32	49	0	1	1
prepare(Map, TopologyContext, OutputCollector)	40%	25%	6	7	11	20	0	1		
execute(Tuple)	54%	50%	n/a	1	1	3	3	1	1	1
declareOutputFields(OutputFieldsDeclarer)	0%	n/a	1	1	3	3	1	1		
getTupleType(Tuple)	36%	25%	4	5	4	7	0	1		
singleSourceArgs(String, CoordinatedBolt.SourceArgs)	0%	n/a	1	1	2	2	1	1		
CoordinatedBolt(IRichBolt, String, CoordinatedBolt.SourceArgs, CoordinatedBolt.IdStreamSpec)	0%	n/a	1	1	1	1	1	1		
getComponentConfiguration()	0%	n/a	1	1	0	1	0	3	0	1
CoordinatedBolt(IRichBolt, Map, CoordinatedBolt.IdStreamSpec)	100%	50%	1	2	0	8	0	1		
cleanup()	100%	n/a	0	1	0	2	0	1		
CoordinatedBolt(IRichBolt)	100%	n/a	0	1	0	1	0	1		
static (...)	100%	n/a	0	1	0	1	0	1		
Total	415 of 573	27%	62 of 72	13%	41	48	83	125	5	12

Figura 28

```

77.     @Override
78.     public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
79.         TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
80.         if (delegate instanceof TimeoutCallback) {
81.             callback = new TimeoutItems();
82.         }
83.         tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
84.         this.collector = collector;
85.         delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
86.         for (String component : Utils.get(context.getThisTargets(),
87.                                         Constants.COORDINATED_STREAM_ID,
88.                                         new HashMap<String, Grouping>())
89.                                         .keySet()) {
90.             for (Integer task : context.getComponentTasks(component)) {
91.                 countOutTasks.add(task);
92.             }
93.         }
94.         if (!sourceArgs.isEmpty()) {
95.             numSourceReports = 0;
96.             for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
97.                 if (entry.getValue().singleCount) {
98.                     numSourceReports += 1;
99.                 } else {
100.                     numSourceReports += context.getComponentTasks(entry.getKey()).size();
101.                }
102.            }
103.        }
104.    }

```

Figura 29

CoordinatedBolt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkFinishId(Tuple, CoordinatedBolt.TupleType)	28%	26%	0%	20	22	32	49	0	1	1
prepare(Map, TopologyContext, OutputCollector)	81%	75%	n/a	2	7	4	20	0	1	1
declareOutputFields(OutputFieldsDeclarer)	0%	n/a	1	1	3	3	1	1		
getComponentConfiguration()	0%	n/a	1	1	1	1	1	1		
execute(Tuple)	100%	87%	1	5	0	27	0	1		
CoordinatedBolt(IRichBolt, Map, CoordinatedBolt.IdStreamSpec)	100%	100%	0	2	0	8	0	1		
getTupleType(Tuple)	100%	87%	1	5	0	7	0	1		
singleSourceArgs(String, CoordinatedBolt.SourceArgs)	100%	n/a	0	1	0	3	0	1		
CoordinatedBolt(IRichBolt, String, CoordinatedBolt.SourceArgs, CoordinatedBolt.IdStreamSpec)	100%	n/a	0	1	0	2	0	1		
cleanup()	100%	n/a	0	1	0	3	0	1		
CoordinatedBolt(IRichBolt)	100%	n/a	0	1	0	2	0	1		
static (...)	100%	n/a	0	1	0	1	0	1		
Total	216 of 573	62%	36 of 72	50%	26	48	40	126	2	12

Figura 30

```

77.     @Override
78.     public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
79.         TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
80.         if (delegate instanceof TimeoutCallback) {
81.             callback = new TimeoutItems();
82.         }
83.         tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
84.         this.collector = collector;
85.         delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
86.         for (String component : Utils.get(context.getThisTargets(),
87.                                         Constants.COORDINATED_STREAM_ID,
88.                                         new HashMap<String, Grouping>())
89.                                         .keySet()) {
90.             for (Integer task : context.getComponentTasks(component)) {
91.                 countOutTasks.add(task);
92.             }
93.         }
94.         if (!sourceArgs.isEmpty()) {
95.             numSourceReports = 0;
96.             for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
97.                 if (entry.getValue().singleCount) {
98.                     numSourceReports += 1;
99.                 } else {
100.                     numSourceReports += context.getComponentTasks(entry.getKey()).size();
101.                }
102.            }
103.        }
104.    }

```

Figura 31

```

169.     @Override
170.     public void execute(Tuple tuple) {
171.         Object id = tuple.getValue(0);
172.         TrackingInfo track;
173.         TupleType type = getTupleType(tuple);
174.         synchronized (tracked) {
175.             track = tracked.get(id);
176.             if (track == null) {
177.                 track = new TrackingInfo();
178.                 if (idStreamSpec == null) {
179.                     track.receivedId = true;
180.                 }
181.                 tracked.put(id, track);
182.             }
183.         }
184.         if (type == TupleType.ID) {
185.             synchronized (tracked) {
186.                 track.receivedId = true;
187.             }
188.             checkFinishId(tuple, type);
189.         } else if (type == TupleType.COORD) {
190.             int count = (Integer) tuple.getValue(1);
191.             synchronized (tracked) {
192.                 track.reportCount++;
193.                 track.expectedTupleCount += count;
194.             }
195.             checkFinishId(tuple, type);
196.         } else {
197.             synchronized (tracked) {
198.                 delegate.execute(tuple);
199.             }
200.         }
201.     }
202. }

```

Figura 32

```

221.     private TupleType getTupleType(Tuple tuple) {
222.         if (idStreamSpec != null
223.             && tuple.getSourceGlobalStreamId().equals(idStreamSpec.id)) {
224.             return TupleType.ID;
225.         } else if (!sourceArgs.isEmpty()
226.             && tuple.getSourceStreamId().equals(Constants.COORDINATED_STREAM_ID)) {
227.             return TupleType.COORD;
228.         } else {
229.             return TupleType.REGULAR;
230.         }
231.     }
232.
233.     enum TupleType {
234.         REGULAR,
235.         ID,
236.         COORD
237.     }

```

Figura 33

```

77
78     @Override
79     public void prepare(Map<String, Object> config, TopologyContext context, OutputCollector collector) {
80     TimeCacheMap.ExpiredCallback<Object, TrackingInfo> callback = null;
81     1     if (delegate instanceof TimeoutCallback) {
82         callback = new TimeoutItems();
83     }
84     tracked = new TimeCacheMap<>(context.maxTopologyMessageTimeout(), callback);
85     1     this.collector = collector;
86     delegate.prepare(config, context, new OutputCollector(new CoordinatedOutputCollector(collector)));
87     for (String component : Utils.get(context.getThisTargets(),
88                                         Constants.COORDINATED_STREAM_ID,
89                                         new HashMap<String, Grouping>())
90                                         .keySet()) {
91         for (Integer task : context.getComponentTasks(component)) {
92             countOutTasks.add(task);
93         }
94     1     if (!sourceArgs.isEmpty()) {
95         numSourceReports = 0;
96         for (Entry<String, SourceArgs> entry : sourceArgs.entrySet()) {
97     1         if (entry.getValue().singleCount) {
98             1             numSourceReports += 1;
99         } else {
100    1             numSourceReports += context.getComponentTasks(entry.getKey()).size();
101         }
102     }
103   }
104 }

```

Figura 34

```

169     @Override
170     public void execute(Tuple tuple) {
171         Object id = tuple.getValue(0);
172         TrackingInfo track;
173         TupleType type = getTupleType(tuple);
174         synchronized (tracked) {
175             track = tracked.get(id);
176             if (track == null) {
177                 track = new TrackingInfo();
178                 if (idStreamSpec == null) {
179                     track.receivedId = true;
180                 }
181                 tracked.put(id, track);
182             }
183         }
184         if (type == TupleType.ID) {
185             synchronized (tracked) {
186                 track.receivedId = true;
187             }
188             checkFinishId(tuple, type);
189         } else if (type == TupleType.COORD) {
190             int count = (Integer) tuple.getValue(1);
191             synchronized (tracked) {
192                 track.reportCount++;
193                 track.expectedTupleCount += count;
194             }
195             checkFinishId(tuple, type);
196         } else {
197             synchronized (tracked) {
198                 delegate.execute(tuple);
199             }
200         }
201     }
202 }
```

Figura 35

```

221     private TupleType getTupleType(Tuple tuple) {
222         if (idStreamSpec != null
223             && tuple.getSourceGlobalStreamId().equals(idStreamSpec.id)) {
224             return TupleType.ID;
225         } else if (!sourceArgs.isEmpty()
226             && tuple.getSourceStreamId().equals(Constants.COORDINATED_STREAM_ID)) {
227             return TupleType.COORD;
228         } else {
229             return TupleType.REGULAR;
230         }
231     }
```

Figura 36

```

123.     protected void validate(Map<String, Object> topoConf, Count windowLengthCount, Duration windowLengthDuration,
124.                             Count slidingIntervalCount, Duration slidingIntervalDuration) {
125.
126.         int topologyTimeout = getTopologyTimeoutMillis(topoConf);
127.         int maxSpoutPending = getMaxSpoutPending(topoConf);
128.         if (windowLengthCount == null && windowLengthDuration == null) {
129.             throw new IllegalArgumentException("Window length is not specified");
130.         }
131.
132.         if (windowLengthDuration != null && slidingIntervalDuration != null) {
133.             ensureDurationLessThanTimeout(windowLengthDuration.value + slidingIntervalDuration.value, topologyTimeout);
134.         } else if (windowLengthDuration != null) {
135.             ensureDurationLessThanTimeout(windowLengthDuration.value, topologyTimeout);
136.         } else if (slidingIntervalDuration != null) {
137.             ensureDurationLessThanTimeout(slidingIntervalDuration.value, topologyTimeout);
138.         }
139.
140.         if (windowLengthCount != null && slidingIntervalCount != null) {
141.             ensureCountLessThanMaxPending(windowLengthCount.value + slidingIntervalCount.value, maxSpoutPending);
142.         } else if (windowLengthCount != null) {
143.             ensureCountLessThanMaxPending(windowLengthCount.value, maxSpoutPending);
144.         } else if (slidingIntervalCount != null) {
145.             ensureCountLessThanMaxPending(slidingIntervalCount.value, maxSpoutPending);
146.         }
147.     }
```

Figura 37

```

123     protected void validate(Map<String, Object> topoConf, Count windowLengthCount, Duration windowLengthDuration,
124                             Count slidingIntervalCount, Duration slidingIntervalDuration) {
125
126         int topologyTimeout = getTopologyTimeoutMillis(topoConf);
127         int maxSpoutPending = getMaxSpoutPending(topoConf);
128         if (windowLengthCount == null && windowLengthDuration == null) {
129             throw new IllegalArgumentException("Window length is not specified");
130         }
131
132         if (windowLengthDuration != null && slidingIntervalDuration != null) {
133             ensureDurationLessThanTimeout(windowLengthDuration.value + slidingIntervalDuration.value, topologyTimeout);
134         } else if (windowLengthDuration != null) {
135             ensureDurationLessThanTimeout(windowLengthDuration.value, topologyTimeout);
136         } else if (slidingIntervalDuration != null) {
137             ensureDurationLessThanTimeout(slidingIntervalDuration.value, topologyTimeout);
138         }
139
140         if (windowLengthCount != null && slidingIntervalCount != null) {
141             ensureCountLessThanMaxPending(windowLengthCount.value + slidingIntervalCount.value, maxSpoutPending);
142         } else if (windowLengthCount != null) {
143             ensureCountLessThanMaxPending(windowLengthCount.value, maxSpoutPending);
144         } else if (slidingIntervalCount != null) {
145             ensureCountLessThanMaxPending(slidingIntervalCount.value, maxSpoutPending);
146         }
147     }

```

Figura 38

```

@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 39

```

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

```

Figura 40

```

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

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

```

Figura 41

```

        @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 42

```

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 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

```

<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>
<counter type="DU" missed="2" covered="32"/>
<counter type="METHOD" missed="0" covered="1"/>
<counter type="CLASS" missed="0" covered="1"/>
</class>

```

Figura 45

```

<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 46

```

<class name="org/apache/bookkeeper/bookie/storage/ldb/WriteCache">
<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.maxSegmentsSize" def="132" use="160" 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>
<method name="get" desc="(J)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="this.allocator" 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 47

```

<method name="initWindowManager" desc="(Long/apache/storm/windowing/WindowManager.java:115)" covered="1">
<du var="this" def="152" use="178" target="180" covered="1"/>
<du var="this" def="152" use="178" target="203" covered="1"/>
<du var="this" def="152" use="208" covered="1"/>
<du var="this" def="152" use="210" covered="1"/>
<du var="this" def="152" use="211" covered="1"/>
<du var="this" def="152" use="213" covered="1"/>
<du var="this" def="152" use="214" covered="1"/>
<du var="this" def="152" use="180" covered="1"/>
<du var="this" def="152" use="181" target="182" covered="1"/>
<du var="this" def="152" use="181" target="188" covered="1"/>
<du var="this" def="152" use="191" covered="1"/>
<du var="this" def="152" use="201" covered="1"/>
<du var="this" def="152" use="201" covered="1"/>
<du var="this" def="152" use="189" covered="1"/>
<du var="this" def="152" use="182" target="183" covered="1"/>
<du var="this" def="152" use="182" target="188" covered="0"/>
<du var="this" def="152" use="164" covered="1"/>
<du var="lifecycleListener" def="152" use="152" covered="1"/>
<du var="lifecycleListener" def="152" use="152" covered="0"/>
<du var="topoConf" def="152" use="160" target="161" covered="1"/>
<du var="topoConf" def="152" use="160" target="162" covered="1"/>
<du var="topoConf" def="152" use="162" target="163" covered="1"/>
<du var="topoConf" def="152" use="162" target="168" covered="0"/>
<du var="topoConf" def="152" use="168" target="169" covered="1"/>
<du var="topoConf" def="152" use="168" target="170" covered="1"/>
<du var="topoConf" def="152" use="170" target="171" covered="1"/>
<du var="topoConf" def="152" use="170" target="175" covered="1"/>
<du var="topoConf" def="152" use="203" target="204" covered="1"/>
<du var="topoConf" def="152" use="203" target="208" covered="1"/>
<du var="topoConf" def="152" use="208" covered="1"/>
<du var="topoConf" def="152" use="180" covered="1"/>
<du var="topoConf" def="152" use="188" target="189" covered="1"/>
<du var="topoConf" def="152" use="188" target="191" covered="1"/>
<du var="topoConf" def="152" use="195" target="196" covered="1"/>
<du var="topoConf" def="152" use="195" target="198" covered="1"/>
<du var="topoConf" def="152" use="196" covered="1"/>
<du var="topoConf" def="152" use="189" covered="1"/>
<du var="topoConf" def="152" use="172" covered="1"/>
<du var="topoConf" def="152" use="169" covered="1"/>
<du var="topoConf" def="152" use="164" covered="1"/>
<du var="topoConf" def="152" use="161" covered="1"/>
<du var="context" def="152" use="201" covered="1"/>
<du var="context" def="152" use="182" target="183" covered="1"/>
<du var="context" def="152" use="182" target="188" covered="0"/>
<du var="queue" def="152" use="152" covered="1"/>
<du var="queue" def="152" use="152" covered="0"/>
<du var="stateful" def="152" use="152" targets="152" covered="0"/>
<du var="stateful" def="152" use="152" target="152" covered="1"/>
<du var="MILLISECONDS" def="152" use="172" covered="0"/>
<du var="MILLISECONDS" def="152" use="164" covered="1"/>
<du var="this.windowLengthDuration" def="152" use="208" covered="1"/>
<du var="this.windowLengthDuration" def="152" use="210" covered="1"/>
<du var="this.timestampExtractor" def="152" use="178" target="180" covered="1"/>
<du var="this.timestampExtractor" def="152" use="178" target="203" covered="1"/>
<du var="manager" def="152" use="211" covered="1"/>
<du var="manager" def="152" use="213" covered="1"/>
<du var="manager" def="152" use="214" covered="1"/>
<du var="manager" def="152" use="215" covered="1"/>
<du var="manager" def="152" use="201" covered="1"/>
<du var="windowLengthCount" def="156" use="208" covered="1"/>
<du var="windowLengthCount" def="156" use="210" covered="1"/>
<du var="slidingIntervalDuration" def="157" use="208" covered="1"/>
<du var="slidingIntervalDuration" def="157" use="211" covered="1"/>
<du var="slidingIntervalCount" def="158" use="208" covered="1"/>
<du var="slidingIntervalCount" def="158" use="211" covered="1"/>
<du var="windowLengthCount" def="161" use="208" covered="1"/>
<du var="windowLengthCount" def="161" use="210" covered="1"/>
<du var="this.windowLengthDuration" def="164" use="208" covered="1"/>
<du var="this.windowLengthDuration" def="164" use="210" covered="1"/>
<du var="slidingIntervalCount" def="169" use="208" covered="1"/>
<du var="slidingIntervalCount" def="169" use="211" covered="1"/>
<du var="slidingIntervalDuration" def="172" use="208" covered="1"/>
<du var="slidingIntervalDuration" def="172" use="211" covered="1"/>
<du var="slidingIntervalCount" def="175" use="208" covered="1"/>
<du var="slidingIntervalCount" def="175" use="211" covered="1"/>
<du var="this.lateTupleStream" def="180" use="181" target="182" covered="1"/>
<du var="this.lateTupleStream" def="180" use="181" target="188" covered="1"/>
<du var="this.lateTupleStream" def="180" use="182" target="183" covered="1"/>
<du var="this.lateTupleStream" def="180" use="182" target="188" covered="0"/>
<du var="this.maxLagMs" def="189" use="201" covered="1"/>
<du var="this.maxLagMs" def="191" use="201" covered="1"/>
<du var="watermarkInterval" def="196" use="201" covered="1"/>
<du var="watermarkInterval" def="198" use="201" covered="1"/>
<counter type="DU" missed="7" covered="76"/>
<counter type="METHOD" missed="0" covered="1"/>

```

Figura 48

```

<method name="execute" desc="(Lorg/apache/storm/tuple/Tuple;)V">
<du var="this" def="307" use="307" target="308" covered="1"/>
<du var="this" def="307" use="307" target="320" covered="1"/>
<du var="this" def="307" use="320" covered="1"/>
<du var="this" def="307" use="308" covered="1"/>
<du var="this" def="307" use="309" target="310" covered="1"/>
<du var="this" def="307" use="309" target="312" covered="0"/>
<du var="this" def="307" use="312" target="313" covered="0"/>
<du var="this" def="307" use="312" target="315" covered="0"/>
<du var="this" def="307" use="317" covered="0"/>
<du var="this" def="307" use="313" covered="0"/>
<du var="this" def="307" use="310" covered="1"/>
<du var="input" def="307" use="320" covered="1"/>
<du var="input" def="307" use="308" covered="1"/>
<du var="input" def="307" use="309" target="310" covered="1"/>
<du var="input" def="307" use="309" target="312" covered="0"/>
<du var="input" def="307" use="315" covered="0"/>
<du var="input" def="307" use="317" covered="0"/>
<du var="input" def="307" use="313" covered="0"/>
<du var="input" def="307" use="313" covered="0"/>
<du var="input" def="307" use="310" covered="1"/>
<du var="this.timestampExtractor" def="307" use="308" covered="1"/>
<du var="this.waterMarkEventGenerator" def="307" use="309" target="310" covered="1"/>
<du var="this.waterMarkEventGenerator" def="307" use="309" target="312" covered="0"/>
<du var="this.windowManager" def="307" use="320" covered="1"/>
<du var="this.windowManager" def="307" use="310" covered="1"/>
<du var="this.lateTupleStream" def="307" use="312" target="313" covered="0"/>
<du var="this.lateTupleStream" def="307" use="312" target="315" covered="0"/>
<du var="this.lateTupleStream" def="307" use="313" covered="0"/>
<du var="this.windowedOutputCollector" def="307" use="317" covered="0"/>
<du var="this.windowedOutputCollector" def="307" use="313" covered="0"/>
<du var="LOG" def="307" use="315" covered="0"/>
<du var="ts" def="308" use="309" target="310" covered="1"/>
<du var="ts" def="308" use="309" target="312" covered="0"/>
<du var="ts" def="308" use="315" covered="0"/>
<du var="ts" def="308" use="310" covered="1"/>
<counter type="DU" missed="19" covered="16"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 49

```

<method name="validate" desc="(Ljava/util/Map;Lorg/apache/storm/topology/base;
<du var="this" def="126" use="145" covered="1"/>
<du var="this" def="126" use="143" covered="0"/>
<du var="this" def="126" use="141" covered="1"/>
<du var="this" def="126" use="137" covered="0"/>
<du var="this" def="126" use="135" covered="1"/>
<du var="this" def="126" use="133" covered="1"/>
<du var="windowLengthCount" def="126" use="128" target="128" covered="1"/>
<du var="windowLengthCount" def="126" use="128" target="132" covered="1"/>
<du var="windowLengthCount" def="126" use="140" target="140" covered="1"/>
<du var="windowLengthCount" def="126" use="140" target="142" covered="1"/>
<du var="windowLengthCount" def="126" use="142" target="143" covered="0"/>
<du var="windowLengthCount" def="126" use="142" target="144" covered="1"/>
<du var="windowLengthCount" def="126" use="143" covered="0"/>
<du var="windowLengthCount" def="126" use="141" covered="1"/>
<du var="windowLengthDuration" def="126" use="132" target="132" covered="1"/>
<du var="windowLengthDuration" def="126" use="132" target="134" covered="1"/>
<du var="windowLengthDuration" def="126" use="134" target="135" covered="1"/>
<du var="windowLengthDuration" def="126" use="134" target="136" covered="1"/>
<du var="windowLengthDuration" def="126" use="135" covered="1"/>
<du var="windowLengthDuration" def="126" use="133" covered="1"/>
<du var="windowLengthDuration" def="126" use="128" target="129" covered="0"/>
<du var="windowLengthDuration" def="126" use="128" target="132" covered="1"/>
<du var="slidingIntervalCount" def="126" use="144" target="145" covered="1"/>
<du var="slidingIntervalCount" def="126" use="144" target="147" covered="1"/>
<du var="slidingIntervalCount" def="126" use="145" covered="1"/>
<du var="slidingIntervalCount" def="126" use="140" target="141" covered="1"/>
<du var="slidingIntervalCount" def="126" use="140" target="142" covered="0"/>
<du var="slidingIntervalCount" def="126" use="141" covered="1"/>
<du var="slidingIntervalDuration" def="126" use="136" target="137" covered="0"/>
<du var="slidingIntervalDuration" def="126" use="136" target="140" covered="1"/>
<du var="slidingIntervalDuration" def="126" use="137" covered="0"/>
<du var="slidingIntervalDuration" def="126" use="132" target="133" covered="1"/>
<du var="slidingIntervalDuration" def="126" use="132" target="134" covered="1"/>
<du var="slidingIntervalDuration" def="126" use="133" covered="1"/>
<du var="windowLengthDuration.value" def="126" use="135" covered="1"/>
<du var="windowLengthDuration.value" def="126" use="133" covered="1"/>
<du var="slidingIntervalDuration.value" def="126" use="137" covered="0"/>
<du var="slidingIntervalDuration.value" def="126" use="133" covered="1"/>
<du var="windowLengthCount.value" def="126" use="143" covered="0"/>
<du var="windowLengthCount.value" def="126" use="141" covered="1"/>
<du var="slidingIntervalCount.value" def="126" use="145" covered="1"/>
<du var="slidingIntervalCount.value" def="126" use="141" covered="1"/>
<du var="topologyTimeout" def="126" use="137" covered="0"/>
<du var="topologyTimeout" def="126" use="135" covered="1"/>
<du var="topologyTimeout" def="126" use="133" covered="1"/>
<du var="maxSpoutPending" def="127" use="145" covered="1"/>
<du var="maxSpoutPending" def="127" use="143" covered="0"/>
<du var="maxSpoutPending" def="127" use="141" covered="1"/>
<counter type="DU" missed="12" covered="36"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 50

```

<method name="prepare" desc="(Ljava/util/Map;Lorg/apache/storm/task/
<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 51

```

<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 52

```

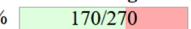
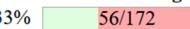
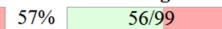
<method name="getTupleType" desc="(Long/apache/storm/tuple/Tuple;)Long/apache/
<du var="this" def="222" use="222" target="222" covered="1"/>
<du var="this" def="222" use="222" target="225" covered="1"/>
<du var="this" def="222" use="225" target="225" covered="1"/>
<du var="this" def="222" use="225" target="229" 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 53

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
3	63%  170/270	33%  56/172	57%  56/99

Breakdown by Package

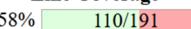
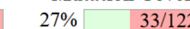
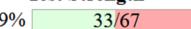
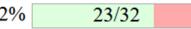
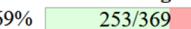
Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
org.apache.bookkeeper.bookie.storage.lmdb	2	58%  110/191	27%  33/122	49%  33/67
org.apache.bookkeeper.util	1	76%  60/79	46%  23/50	72%  23/32

Figura 54

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	69%  253/369	37%  65/177	53%  65/123

Breakdown by Package

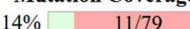
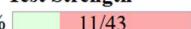
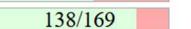
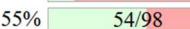
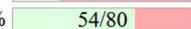
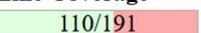
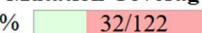
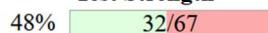
Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
org.apache.storm.coordination	1	58%  115/200	14%  11/79	26%  11/43
org.apache.storm.topology	1	82%  138/169	55%  54/98	68%  54/80

Figura 55

Pit Test Coverage Report

Package Summary

org.apache.bookkeeper.bookie.storage.ldb

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	58%  110/191	26%  32/122	48%  32/67

Breakdown by Class

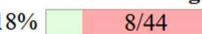
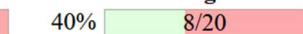
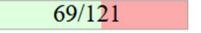
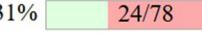
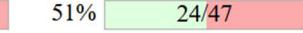
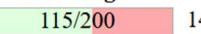
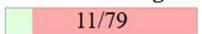
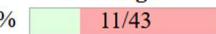
Name	Line Coverage	Mutation Coverage	Test Strength
ReadCache.java	59%  41/70	18%  8/44	40%  8/20
WriteCache.java	57%  69/121	31%  24/78	51%  24/47

Figura 56

Pit Test Coverage Report

Package Summary

org.apache.storm.coordination

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	58%  115/200	14%  11/79	26%  11/43

Breakdown by Class

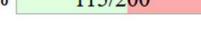
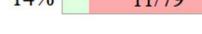
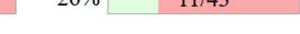
Name	Line Coverage	Mutation Coverage	Test Strength
CoordinatedBolt.java	58%  115/200	14%  11/79	26%  11/43

Figura 57

Pit Test Coverage Report

Package Summary

org.apache.storm.topology

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	82%  137/167	55%  54/98	68%  54/80

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
WindowedBoltExecutor.java	82%  137/167	55%  54/98	68%  54/80

Figura 58