Pasquale Caporaso
0286761
caporasopasquale97@gmail.com

Report ISPW2
Modulo Software Testing

2021-02-08

1 Introduzione

Le attività di testing rappresentano molto spesso uno dei passi più delicati del processo di sviluppo di un software, in questo report si vuole presentare l'attività di testing svolta su due grandi applicazioni Open-Source prodotte dalla Apache Software Foundation, BookKeeper e ZooKeeper. Cercherò di descrivere l'intero processo che ha portato alla creazione e al miglioramento dei vari test effettuati, mi sono attenuto alle tecniche studiate durante il corso e ho applicato il concetto di Continuous Integration integrando i test automticamente nella fase di build del progetto.

2 Scelta delle classi

La scelta delle classi da testare è stata principalmente influenzata dai risultati ottenuti dallo studio dei classificatori effettuato nell'altro modulo di questo corso, ho utilizzato il classificatore con le migliori prestazioni e lo utilizzato per fare una predizione delle classi "Buggy" presenti nell'ultima versione delle applicazioni. Dopo aver fatto ciò ho prelevato dalla lista le classi che presentavano qualche anomalia statistica o che contenevano dei metodi il cui testing risultava più "interessante", ovvero i metodi che permettevano di applicare in maniera più completa le tecniche studiate a lezione. Ho cercato di evitare una scelta basata troppo sui valori numerici delle metriche calcolate in quanto non so in che modo esse vengono utilizzate dal classificatore. Detto ciò le classi e i metodi scelti sono elencati qui sotto:

2.1 BookKeeper

BookKeeper.java
La classe BookKeeper è stata scelta per la sua centralità all'interno del progetto in quanto rappresenta il Client per l'interazione con il sistema, i metodi su cui è stato effetuato testing sono i seguenti:

- CreateLedger, utilizzato per la creazione di un ledger
- asyncDeleteLedger, utilizzato per l'eliminazione di un ledger
- OpenLedger, utilizzato per aprire un ledger e preparare le operazioni di scrittura

Bookie.java
La classe Bookie è stata scelta perchè presentava un valore relativamente alto per la metrica ChgSet e perchè presentava alcuni metodi che sembravano interessanti. I metodi scelti per il testing sono i seguenti:

- addEntry, utilizzato per aggiungere una entry a un ledger
- readEntry, utilizzato per leggere un entry di un ledger

2.2 ZooKeeper

ObserverMaster.java
La classe ObserverMaster è stata scelta per la sua grandezza e perchè presentava alcuni metodi influenzati significativamente dallo stato interno della classe, cosa che ha aggiunto alcune sfide al momento del testing. I metodi scelti sono i seguenti:

- cacheCommittedPacket, utilizzato per aggiungere un pacchetto in stato committed nella coda
- removeProposedPacket, utilizzato per rimuovere un pacchetto in stato proposed dalla coda
- startForwarding, funzione che comunica ad un Observer tutti i pacchetti committati che non ha ancora visto

ZookeeperMain.java
La classe ZookeeperMain è stata scelta perchè rappresenta un punto di interazione tra l'utente e il sistema, è importante quindi che la classe funzioni correttamente per evitare di generare confusione per gli utilizzatori del sistema. I metodi scelti per il testing sono i seguenti:

- parseCommand, utilizzato per interpretare un comando inserito dall'utente
- parseOptions, utilizzato per interpretare i parametri che l'utente ha inserito

3 Domain partitioning

Applicando le tecniche utilizzate a lezione ho separato il dominio di input delle funzioni nelle classi di equivalenza presentate in questa sezione.

3.1 BookKeeper

BookKepper.createLedger

I parametri del metodo sono:

- ensSize (int), il numero di nodi in cui il ledger è salvato
- writeQuorumSize (int), il numero di nodi in cui un entry può essere scritta
- ackQuorumSize (int), il numero di nodi necessari per accettare un operazione sul ledger
- digestType (DigestType), tipo di *redundancy check* da applicare alle entry del ledger
- password (byte[]), un password per il ledger
- customMetadata (Map -String, byte[] -), informazioni aggiuntive riguardanti il ledger

Le classi per i parametri digestType, password e customMeta sono relativamente semplici da individuare, in quanto il primo è una semplice enum, il secondo una stringa e il terzo una mappa, tutti e tre senza particolari restrizioni.
I primi tre parametri invece hanno bisogno di un studio un po' più approfondito, sulla documentazione è presente la seguente relazione che deve essere mantenuta tra essi:

$$ensSize >= writeQuorumSize >= ackQuorumSize$$

Inoltre dal significato che questi parametri hanno all'interno del sistema si evincono le seguenti relazioni:

$$ensSize >= writeQuorumSize > 0$$
$$ackQuorumSize >= 0$$

$$ackQuorumSize <= writeQuorumSize <= ensSize < \#nodinelsistema$$

Questo perchè il ledger deve essere scritto in almeno un nodo e non possiamo scrivere un ledger o pretendere di ricevere un ack da un numero di nodi maggiore rispetto al totale di nodi nel sistema. Mi aspetto, quindi, che l'operazione di creazione abbia successo SOLO SE tutte le relazioni vengono rispettate, detto ciò le classi di equivalenze scelte sono le seguenti:

ensSize: $\{\leq 0\}, \{> 0, \leq totnodi\}, \{> totnodi\}$
writeQuorumSize: $\{\leq 0\}, \{> 0, \leq ensSize \leq totnodi\}, \{> 0, > ensSize, \leq totnodi\}, \{> totnodi\}$
ackQuorumSize: $\{< 0\}$,$\{> 0, \leq writeQuorumSize \leq totnodi\}, \{> 0, > writeQuorumSize, \leq totnodi\}, \{> totnodi\}$
digestType: {MAC}, {CRC32}, {CRC32C}, { DUMMY}
password: { empty_string } , { random_string }
customMetaData: {null}, {empty_map}, {some_metadata}

Infine, gli ultimi tre parametri possono essere testati in modo unidimensionale, mentre per i primi tre, vista la dipendenza, sarebbe utile procedere in maniera multidimensionale, tuttavia, facendo il prodotto cartesiano otteniamo un numero troppo alto di combinazioni. Invece di usare tutte le possibili combinazioni ho deciso quindi di prenderne un sottoinsieme in modo da ottenere almeno un caso di test per ognuna delle relazioni esposte sopra.

BookKepper.openLedger

I parametri del metodo sono:

- lId (long), id del ledger che vogliamo aprire
- digestType (DigestType), tipo di *redundancy check* di questo ledger
- password (byte[]), la password del ledger che vogliamo aprire

I parametri password e digestType sono gli stessi del metodo precedente, per quanto riguarda lId sembra che il sistema utilizzi qualsiasi valore numerico. Mi aspetto che l'operazione fallisca se scelgo un id che non corrisponde a nessun ledger del sistema o in caso la password che ho inserito sia sbagliata. Le partizioni scelte sono quindi:

lId: {existing_ledger_id}, {non_existing_ledger_id}
digestType: {MAC}, {CRC32}, {CRC32C}, { DUMMY}
password: { empty_string } , { correct_password } , { bad_password }

Tutti questi parametri possono essere testati unidimensionalmente facendo attenzione ai casi di test che falliscono per una password non corretta o per un ledger non esistente.

BookKepper.asyncDeleteLedger

I parametri del metodo sono:

- lId (long), id del ledger che vogliamo cancellare
- cb (DeleteCallback), classe di callback chiamata alla fine dell'operazione
- ctx (Object), oggetto di controllo passato al CallBack

Il parametro lId e lo stesso trattato in precedenza. Il secondo parametro è un interfaccia quindi ho deciso di creare dei casi di test con una mia implementazione e dei casi con un'implementazione già presente nell'applicazione. Il terzo parametro è un oggetto qualsiasi e dipende dall'implementazione del CallBack. Le classi di equivalenza scelte per questi parametri sono:

lId: {existing_ledger_id}, {non_existing_ledger_id}
cb: {null}, {my_callback}, {apache_callback}
ctx: { null } , { valid_object }

I parametri possono essere testati unidimensionalmente, facendo però attenzione ad associare il CallBack con un oggetto di controllo corretto.

Bookie.addEntry

I parametri del metodo sono:

- entry (ByteBuf), buffer che contiene il ledger_id, l'entry_id e l'informazione che vogliamo salvare nella entry, questo è un parametro composto quindi creerò partizioni per ognuna delle sue componenti
- ackBeforeSync (boolean), se volgiamo o meno aspettare un ack dal sistema prima di uscire dal metodo
- cb (WriteCallback), classe di callback chiamata alla fine dell'operazione
- ctx (Object), oggetto di controllo passato al CallBack

- masterKey (byte[]), chiave per la entry

Per quanto riguarda i parametri *cb* e *ctx* procedo come per il metodo *asyncDeleteLedger*, la master key è una semplice sequenza di byte e l'ackBeforeSync è un booleano quindi le partizioni sono relativamente semplici. Per quanto riguarda il parametro entry è meglio considerare le sue componenti: il ledger_id può essere trattato come nei precedenti metodi, l'entry_id, come da documentazione, deve essere maggiore di 0 e, infine, per le informazioni che verranno salvate nella entry dovrebbe essere sufficente considerare il caso di entry vuota e quello di entry non vuota. Le classi di equivalenza scelte sono quindi:

entry.ledger_id: {existing_ledger_id}, {non_existing_ledger_id}
entry.entry_id: $\{< 0\}$, $\{\geq 0\}$
entry.info: {null}, {valid_object}
cb: {null}, {my_callback}, {apache_callback}
ctx: { null } , { valid_object }
master_key: {empty_string}, {valid_string}

I parametri possono essere testati in maniera unidimensionale con la solita attenzione al CallBack e all'oggetto di controllo.

Bookie.readEntry

I parametri del metodo sono:

- ledgerId (long), id del ledger che vogliamo leggere
- entryId (long), id della entry che vogliamo leggere

Le classi di equivalenza scelte sono:

ledgerId: {existing_ledger_id}, {non_existing_ledger_id}
entryId: {existing_entry_id}, {non_existing_entry_id}

I parametri possono essere studiati in maniera multidimensionale.

3.2 ZooKeeper

ObserverMaster.cacheCommittedPacket

I parametri del metodo sono:

- pkt (QuorumPacket), pacchetto da inserire nella coda dei committed

La coda dei pacchetti committated ha una dimensione massima e il metodo cerca di mantenere libero almeno un suo quinto, se, inserendo il nuovo pacchetto, lo spazio libero diventa di meno, il sistema elimina fino a 5 dei paccehtti più vecchi. Alla fine di questa operazione però la coda può però comunque avere una dimensione superiore ai 4/5 del limite. Alla fine del metodo, però, ci si assicura che il limite superiore non venga superato eliminando tutti i pacchetti necessari partendo dal più vecchio e, se questo non bastasse (nel caso in cui la dimensione del pacchetto supera la dimensione della coda), il nuovo pacchetto non viene inserito, in tal caso il risultato dell'operazione è il completo svuotamento della coda. Le classi di equivalenza scelte sono:

pkt: $\{dim\_iniziale\_coda + dim\_pkt \leq 0.8 * limite\_superiore\}$, $\{0.8 * limite\_superiore < dim\_iniziale\_coda + dim\_pkt \leq limite\_superiore, \}$, $\{dim\_iniziale\_coda + dim\_pkt > limite\_superiore, dim\_pkt < limite\_superiore\}$, $\{dim\_pkt > limite\_superiore\}$

Nota: non è stato possibile creare un test per l'ultimo caso in quanto non sono riuscito a creare un pacchetto sufficentemente grande ne a ridurre il limite massimo della coda.

ObserverMaster.removeProposedPacket

I parametri del metodo sono:

- zxid (long), id del pacchetto da rimuovere

I pacchetti proposti si trovano in una coda quindi l'id del pacchetto deve corrispondere al primo inserito, se così non fosse l'operazione deve fallire. Inoltre è interessante testare cosa succede quando la coda è vuota quindi è utile aggiungere un altro parametro al test, le classi di equivalenza sono quindi:

fill_queue: {true}, {false}
zxid: $\{< first\_pkt\_id\}$, $\{first\_pkt\_id\}$, $\{> first\_pkt\_id\}$

I parametri possono essere studiati in maniera multidimensionale.

ObserverMaster.startForwarding

I parametri del metodo sono:

- learnerHandler (LearnerHandler), oggetto che rappresenta il nodo a cui dobbiamo fare il forward dei pacchetti
- lastSeenZxid (long), id dell'ultimo pacchetto visto dal nodo

Per quanto riguarda il learner handle non sono riuscito a costruire un oggetto della classe che fosse invalido, quindi ho testato solo il caso null e quello con un oggetto valido. Per quando riguarda il secondo parametro il metodo fallisce se il learner ha è rimasto a un pacchetto che non è più presente nella coda. Inoltre, come per il precedente metodo, è interessante studiare dei casi in cui la coda è vuota. Quindi le classi di equivalenza sono queste:

fill_queue: {true}, {false}
learnerHandle: {null}, {valid_object}
lastSeenZxid: $\{< first\_pkt\_id\}$, $\{first\_pkt\_id\}$, $\{> first\_pkt\_id\}$

I parametri potrebbero essere studiati in maniera multidimensionale visto il numero ridotto di combinzioni, tuttavia il metodo fallsice immediatamente se il primo parametro è null quindi non è necessario testare altre combinazioni.

ZooKeeperMain.parseCommand

I parametri del metodo sono:

- cmdString (string), stringa contenente il commando digitato sul terminale

Dalla documentazione, il stringa contenente il comando deve essere nel seguente formato "cmd arg1 arg2..etc", inoltre è possibile circondare il comando o un qualsiasi argomento con un singolo o doppio apice. Non è compito di questo metodo stabilire se questi sono comandi o argomenti validi quindi esso non dovrebbe fallire a meno che non venga passata una stringa nulla o vuota. Le classi di equivalenza scelte per questo metodo sono:

cmdString: {null}, {empty_string}, {strings in "cmd arg1 arg2..etc" format}, {strings with single quoted command}, {strings with single quoted argument}, {strings with double quoted command}, {strings with double quoted argument}

ZooKeeperMain.parseOptions

I parametri del metodo sono:

- args (string[]), vettore contenente il comando e le flag di un comando inserito da terminale

Dalla documentazione, le flag disponibli sono : -server, -timeout, -client-configuration e -r. Le prime 3 flag inoltre hanno bisogno di essere seguite da un'altra stringa che rappresenta l'algomento dell'opzione scelta, in caso non fosse presente il metodo fallisce, per quanto riguarda la flag -r essa non ha bisogno di un argometo ma anche se fosse inserito questo non causa il fallimento del metodo. Infine, il comando può essere una stringa qualsiasi. Le classi di equivalenza scelte sono quindi:

args: {empty_array}, {strings with -server and no argument}, {strings with -timeout and no argument}, {strings with -client-configuration and no argument}, {strings with -r and a argument}, {strings with -server and a argument}, {strings with -timeout and a argument}, {strings with -client-configuration and a argument}, {strings with -r and a argument}, {strings with a command}

4 Risultati dei test e studio della coverage

4.1 BookKeeper

BookKeeper.createLedger

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
// function signature
// LedgerHandle createLedger(int ensSize, int writeQuorumSize, int ackQuorumSize, DigestType
    digestType, byte[] passwd, final Map<String, byte[]> customMetadata

//fail beacuse of negative/0 ensSize
{false, -1 , -1, -1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},

//these should cause error but don't
//technically documentation does not say this value are bad but they don't really make sense
//{false, 4 , -1, -1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
//{false, 0 , 0, 0, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
//{false, 4 , 0, 0, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},

//valid configurations
{true, 4 , 2, 1, BookKeeper.DigestType.MAC, "".getBytes(), validMetadata},
{true, 4 , 2, 1, BookKeeper.DigestType.CRC32, "password".getBytes(), validMetadata},
{true, 4 , 2, 1, BookKeeper.DigestType.CRC32C, "password".getBytes(), null},
{true, 4 , 2, 1, BookKeeper.DigestType.DUMMY, "password".getBytes(), Collections.emptyMap()},
{true, 8 , 2, 1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{true, 8 , 8, 1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{true, 8 , 8, 8, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{true, 4 , 2, -1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{true, 4 , 2, 0, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},

//fail because ensSize, write or ack quorum are bigger than the actual number of live bookies
{false, 10 , 2, 1, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{false, 10 , 10, 2, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
{false, 10 , 10, 10, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},

//should fail because ensSize < writeQuorum but doesn't - documentation prohibits this values
//{false, 4 , 5, 3, BookKeeper.DigestType.MAC, "password".getBytes(), validMetadata},
```

Questo è l'unico metodo in cui ho avuto dei risultati che non mi riesco a spiegare, il metodo fallisce correttamente in alcuni casi ma sembra creare un Ledger "difettoso" nei casi di test che sopra ho commentato. Per Ledger difettoso intendo un ledger che possiede un id e metadata ma che causa un fallimento inaspestato quando provo a eseguire una quasiasi operazione su di esso. Non sono riuscito a risolvere questo problema, la mia ipotesi è che ci sia un bug nel metodo studiato o nella classe di test che crea il setup del sistema, per confermare questa ipotesi però è necessario uno studio più approfondito dell'applicazione.
Per quanto riguarda la coverage, i valori in questo caso non sono corretti in quanto non ho incluso i metodi che falliscono, comunque come mostrato in figura 1 ho ottenuto una statement coverage pari a 81% e una branch coverage pari a 50%. Non sono riuscito a migliorare queste statistiche in quanto il blocco di codice mancante viene raggiunto solo quando il la funzione precedente ritorna una ledgerHandle nullo e non ho trovato nessuna configurazione che ritorna null senza sollevare un eccezione.

BookKeeper.openLedger

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//LedgerHandle openLedger(long lId, DigestType digestType, byte[] passwd)

//fail beacuse of wrong id
{false, 12345, BookKeeper.DigestType.MAC, "password".getBytes(), false},

//fail because of wrong password
{false, 333, BookKeeper.DigestType.MAC, "bad_password".getBytes(), false},
{false, 333, BookKeeper.DigestType.MAC, "".getBytes(), false},

//valid configurations
{true, 333, BookKeeper.DigestType.MAC, "password".getBytes(),false},
{true, 333, BookKeeper.DigestType.CRC32C, "password".getBytes(),false},
{true, 333, BookKeeper.DigestType.CRC32, "password".getBytes(), false},
{true, 333, BookKeeper.DigestType.DUMMY, "password".getBytes(), false},
```

Il metodo si comporta come previsto, con questi casi di test ottengo una statement coverage pari al 100% e una branch coverage non calcolabile, questo succede perchè, come osserviamo in figura 4 il metodo esso non fa altro che chiamare il suo equivalente asincrono e aspettare il risultato. E' quindi più utile osservare la coverage su quest'altro metodo (figura 5), qui abbiamo una statement coverage dell'80% e una branch coverage del 50%, per migliorare queste statistiche è sufficente aggiungere un caso in cui la classe di bookkeeper che stiamo usando è stata chiusa:

```
//fail because closed
{true, 333, BookKeeper.DigestType.MAC, "password".getBytes(), true},
```

Aggiunto questo le due coverage considerate salgono entrambe al 100%.

BookKeeper.asyncDeleteLedger

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//fail because of closed bookkeeper
{false, -12345, cb1, ctx, true},

//Deleting a non existent ledger is considered a successful delete operation
{true, -12345, cb1, ctx, false},
{true, 12345, cb1, ctx, false},
{true, -12345, cb2, null, false},
{true, 12345, cb2, null, false},

//valid configurations
{true, 333, null, ctx, false},
{true, 333, cb1, ctx, false},
{true, 333, cb2, null, false},
```

Il metodo si comporta come previsto anche se non sicuro perchè la cancellazione di un ledger non-esistente è comunque considerata un successo. Anche per questo metodo, come per il precedente, non avevo considerato il caso in cui l'oggetto usato fosse chiuso, quindi inizialmente la statement coverage era 81% e la branch 50% (figura 2), dopo aver aggiunto il caso entrambe sono salite al 100% (figura 3).

Bookie.addEntry

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//bad configurations - entry cannot be null
{false, null, true, cb2, null, "key".getBytes()},
{false, badEntry, true, cb2, null, "key".getBytes()},

//valid configurations
{true, entry3, true, cb2, null, "key".getBytes()},
{true, entry3, false, cb2, null, "key".getBytes()},
{true, entry3, true, cb1, ctx, "key".getBytes()},
{true, entry3, true, cb2, null, "".getBytes()},
{true, entryWithData, true, cb2, null, "".getBytes()},

//valid configuration, a new ledger is created when it does not exists
{true, entry1, true, cb2, null, "key".getBytes()},
{true, entry2, true, cb2, null, "key".getBytes()},
```

Il risultato del test è ragionevole ma, non sicuro del perchè, quando voglio aggiungere una entry a un ledger non esistente, esso viene creato.
La statement coverage di questo metodo è 89% mentre la branch coverage è 75% (figura 7), non sono riuscito a migliore queste metriche in quando non è stato possibile ne impostare a "Fenced" lo stato del ledger, ne creare un bookie con una directory non writable.

Bookie.readEntry

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//bad configurations - wrong ID or wrong entry
{false, 1234, EXISTING_ENTRY_ID},
{false, 333, 1234},

//valid configuration
{false, 333, EXISTING_ENTRY_ID}
```

Il risultato del test è coerente con quello che mi aspettavo.
La statement coverage per il metodo è 81% mentre la branch coverage è 50%, ancora una volta non sono riuscito a migliorarle. L'unico modo per aumentarle era la disabilitazione di un log di sistema, cosa che non sono stato in grado di fare.

4.2 ZooKeeper

ObserverMaster.cacheCommittedPkts

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//void cacheCommittedPacket(final QuorumPacket pkt)

//null packets are not allowed
{false, null},

//valid configurations
{true, undersizedPkt},
{true, almostOverSizedPkt},
{true, overSizedPkt},
{true, reallyOverSizedPkt}
```

Tutti i test si comportano come previsto.
Per quanto riguarda la coverage, inizialmente avevo una statement coverage pari a 39% e una branch coverage

pari a 20%, questi valori erano dovuti a un mio errore nella creazione dei pacchetti, infatti cercavo di generare dei vettori troppo grandi che venivano essenzialmente cancellati durante l'esecuzione.
Risolvendo questo problema la statement coverage è salita al 67% e la branch al 60% (figura 12, le restanti istruzioni o condizioni non esplorate sono dovute al fatto che non sono riuscito a creare pacchetti sufficentemente grandi, ho spiegato meglio questo problema nella sezione 3.2

ObserverMaster.startForwanding

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//long startForwarding(LearnerHandler learnerHandler, long lastSeenZxid)

//invalid configuration - learner is null or invalid
{-2, null, 30, true},

//invalid configuration - learner is too far behind
{-1, validLh, 10, true},

//method does nothing with empty queue, test only one configuration
{0, validLh, 30, false},

//valid configurations
{0, validLh, 30, true},
{0, validLh, FIRST_PKT_ID-1, true},
{0, validLh, FIRST_PKT_ID+10, true},
```

Tutti i test si comportano come previsto.
Inizialmente la statement coverage era 97% e la branch coverage era 90% (figura 13), questo era dovuto ad uno dei casi di test non configurato correttaemente. I paratri del test scritti qui sopra hanno una statement e branch coverage del 100% (figura 14).

ObserverMaster.removeProposedPkt

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//QuorumPacket removeProposedPacket(long zxid) , boolean fillqueue

//invalid configurations
{false, FIRST_PKT_ID-1, true},
{false, FIRST_PKT_ID, false},
{false, FIRST_PKT_ID+1, true},

//valid configuration
{true, FIRST_PKT_ID, true}
```

Tutti i test si comportano come previsto. La statement e branch coverage sono 100%.

ZooKeeperMain.parseCommand

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//boolean parseCommand(String cmdstring)

//invalid configuration
{false, null},
{false, ""},
```

```
//valid configurations? - somehow an empty argument is still valid
{true, "'' arguments"},
{true, "\"\" arguments"},
{true, "cmd ''"},
{true, "cmd \"\""},

//valid configurations
{true, "cmd arg1 arg2"},
{true, "cmd 'arg1' arg2"},
{true, "cmd arg1 \"arg2\""},
{true, "'cmd' arg1 arg2"},
{true, "\"cmd\" arg1 arg2"},
```

Tutti i test si comportano come previsto, anche se mi sorprende che delle virgolette vuote sono una configurazione valida.
La statement e la branch coverage sono entrambe 100% (figura 9).

ZooKeeperMain.parseOptions

I casi di test creati sono i seguenti, il primo parametro di ogni riga rappresenta il risultato che mi aspetto dal test.

```
//function signature
//boolean parseOptions(String[] args)

//invalid configuration - empty options
{false, new String[1]},

//invalid configurations - missing arguments
{false, new String[]{"-server"}},
{false, new String[]{"-timeout"}},
{false, new String[]{"-client-configuration"}},

//valid configurations
{true, new String[]{"-r"}},
{true, new String[]{"-r", "cmd"}}, //correct - the argument can be after the option
{true, new String[]{"-server", "argument"}},
{true, new String[]{"-timeout", "argument"}},
{true, new String[]{"-client-configuration", "argument"}},
{true, new String[]{"cmd", "cdaArg", "-client-configuration", "argument"}},
```

Tutti i test si comportano come previsto.
Inizialmente la statement coverage era 93% e la branch coverage era 92% (figura 10), questo perchè avevo dimenticato si inserire questo caso di test:

```
{true, new String[]{"cmd", "cdaArg", "-client-configuration", "argument"}},
```

Dopo averlo aggiunto entrambe le statistiche vanno al 100% (figura 11).

5 Studio delle mutazioni

5.1 BookKeeper

Bookie

La classe Bookie ha una mutation coverage complessiva del 26%, i singoli metodi hanno i seguenti risultati:

- addEntry: 4 Killed, 3 Survived
- readEntry: 1 Killed, 6 Survived

Il metodo readEntry ha una coverage minore, questo è probabilemente dovuto al minor numero di casi di test.

BookKeeper

La classe BookKeeper ha una mutation coverage complessiva del 33%, i singoli metodi hanno i seguenti risultati:

- createLedger: 3 Killed, 0 Survived
- openLedger: 1 Killed, 1 Timed out
- asyncDeleteLedger: 2 Killed, 3 Timed Out

Nonostante BookKeeper sia più grande di Bookie ho ottenuto complessivamente una coverage maggiore, questo è probabilemente dovuto al numero maggiore di connessioni tra i metodi.

5.2 ZooKeeper

ObserverMaster

La classe ObserverMaster ha una mutation coverage complessiva del 17%, i singoli metodi hanno i seguenti risultati:

- cacheCommittedPacket: 4 Killed, 9 Survived
- openLedger: 1 Killed, 1 Timed out
- asyncDeleteLedger: 2 Killed, 3 Timed Out

La classe ObserverMaster è sicuramente la più complessa tra quelle studiate visto il suo ruolo nel sistema, questo spiega anche il basso livello di mutation coverage. Per migliorare questa statistica sarebbero necessari dei test più approfonditi magari con la simulazione delle interazioni tra i var inodi del sistema.

ZookeeperMain.MyCommandOptions

La classe ZookeeperMain.MyCommandOptions ha una mutation coverage complessiva del 64%, i singoli metodi hanno i seguenti risultati:

- parseCommand: 5 Killed, 1 Survived
- parseOptions: 9 Killed, 2 Survived

La classe ZookeeperMain ha avuto i risultati migliori in termini di Mutation Coverage, questo è dovuto in parte alla sua "semplicità" ma principalmente al fatto che ho dovuto estrarre da essa la classe interna che conteneva i metodi che ho studiato. Infatti PIT non sempra essere in grado di calcolare le mutazioni delle classi interne, l'unica soluzione è stata quella di estrarre la classe e inserirla in un nuovo file.

6 Considerazioni finali

Questo progetto è stato è stato molto complesso, ho avuto molte difficoltà nella corretta configurazione dei tool in quanto online non è presente molto materiale che ne spiega il funzionamento. Per quanto riguarda i risultati dei test, in alcuni casi sono rimasto sorpreso da alcuni risultati, questo è probabilemente dovuto alla mia poca esperienza con grandi applicazioni OpenSource e alla conoscenza comunque limita dei due software studiati. Alla fine di tutto sono relativamente soddisfatto dai risultati in termini di coverage dei miei test anche se, se avessi avuto a disposizione più tempo, mi sarebbe piaciuto migliorarli ulteriormente.

7 Note aggiuntive

Il testing è stato effettuato su zookeeper versione 3.5.8-rc0 e non sull'ultima versione disponibile (3.5.9) in quanto avevo già configurato sonarcloud e travis per il fork personale nel mese di giugno/luglio 2020 e, viste le grosse difficoltà incontrate, non volevo causare problemi aggiornando la repository.
Per quanto riguarda bookkeeper invece è stata usata l'ultima versione disponibile (4.12.0), questo perchè la repository configurata a luglio 2020 aveva problemi con dipendenze non più disponibili. Ho quindi deciso di aggiornare all'ultima versione e configurare di nuovo il tutto.
L'ambiente di sviluppo usato per la creazione dei test di entrambe le applicazioni è IntelliJ su una macchina Linux Ubuntu 18.04 LTS.

8 Figure

```
    /**
     * Synchronous call to create ledger. Parameters match those of asyncCreateLedger
     *
     * @param ensSize
     * @param writeQuorumSize
     * @param ackQuorumSize
     * @param digestType
     * @param passwd
     * @param customMetadata
     * @return a handle to the newly created ledger
     * @throws InterruptedException
     * @throws BKException
     */
    public LedgerHandle createLedger(int ensSize, int writeQuorumSize, int ackQuorumSize,
                                     DigestType digestType, byte[] passwd, final Map<String, byte[]> customMetadata)
            throws InterruptedException, BKException {
        CompletableFuture<LedgerHandle> future = new CompletableFuture<>();
        SyncCreateCallback result = new SyncCreateCallback(future);

        /*
         * Calls asynchronous version
         */
        asyncCreateLedger(ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd,
                          result, null, customMetadata);

        LedgerHandle lh = SyncCallbackUtils.waitForResult(future);
        if (lh == null) {
            LOG.error("Unexpected condition : no ledger handle returned for a success ledger creation");
            throw BKException.create(BKException.Code.UnexpectedConditionException);
        }
        return lh;
    }
```

Figure 1: Create ledger

```java
/**
 * Deletes a ledger asynchronously.
 *
 * @param lId
 *            ledger Id
 * @param cb
 *            deleteCallback implementation
 * @param ctx
 *            optional control object
 */
public void asyncDeleteLedger(final long lId, final DeleteCallback cb, final Object ctx) {
    closeLock.readLock().lock();
    try {
        if (closed) {
            cb.deleteComplete(BKException.Code.ClientClosedException, ctx);
            return;
        }
        new LedgerDeleteOp(BookKeeper.this, clientStats, lId, cb, ctx).initiate();
    } finally {
        closeLock.readLock().unlock();
    }
}
```

Figure 2: Async Delete ledger pre miglioramenti

```java
/**
 * Deletes a ledger asynchronously.
 *
 * @param lId
 *            ledger Id
 * @param cb
 *            deleteCallback implementation
 * @param ctx
 *            optional control object
 */
public void asyncDeleteLedger(final long lId, final DeleteCallback cb, final Object ctx) {
    closeLock.readLock().lock();
    try {
        if (closed) {
            cb.deleteComplete(BKException.Code.ClientClosedException, ctx);
            return;
        }
        new LedgerDeleteOp(BookKeeper.this, clientStats, lId, cb, ctx).initiate();
    } finally {
        closeLock.readLock().unlock();
    }
}


/**
 * Synchronous call to delete a ledger. Parameters match those of
 * {@link #asyncDeleteLedger(long, AsyncCallback.DeleteCallback, Object)}
 *
 * @param lId
 *            ledgerId
 * @throws InterruptedException
 * @throws BKException
 */
public void deleteLedger(long lId) throws InterruptedException, BKException {
    CompletableFuture<Void> future = new CompletableFuture<>();
    SyncDeleteCallback result = new SyncDeleteCallback(future);
    // Call asynchronous version
    asyncDeleteLedger(lId, result, null);

    SyncCallbackUtils.waitForResult(future);
}
```

Figure 3: Async Delete ledger post miglioramenti

```java
/**
 * Synchronous open ledger call.
 *
 * @see #asyncOpenLedger
 * @param lId
 *          ledger identifier
 * @param digestType
 *          digest type, either MAC or CRC32
 * @param passwd
 *          password
 * @return a handle to the open ledger
 * @throws InterruptedException
 * @throws BKException
 */
public LedgerHandle openLedger(long lId, DigestType digestType, byte[] passwd)
        throws BKException, InterruptedException {
    CompletableFuture<LedgerHandle> future = new CompletableFuture<>();
    SyncOpenCallback result = new SyncOpenCallback(future);

    /*
     * Calls async open ledger
     */
    asyncOpenLedger(lId, digestType, passwd, result, null);

    return SyncCallbackUtils.waitForResult(future);
}
```

Figure 4: Open Ledger pre miglioramenti

```java
/**
 * Open existing ledger asynchronously for reading.
 *
 * <p>Opening a ledger with this method invokes fencing and recovery on the ledger
 * if the ledger has not been closed. Fencing will block all other clients from
 * writing to the ledger. Recovery will make sure that the ledger is closed
 * before reading from it.
 *
 * <p>Recovery also makes sure that any entries which reached one bookie, but not a
 * quorum, will be replicated to a quorum of bookies. This occurs in cases were
 * the writer of a ledger crashes after sending a write request to one bookie but
 * before being able to send it to the rest of the bookies in the quorum.
 *
 * <p>If the ledger is already closed, neither fencing nor recovery will be applied.
 *
 * @see LedgerHandle#asyncClose
 *
 * @param lId
 *          ledger identifier
 * @param digestType
 *          digest type, either MAC or CRC32
 * @param passwd
 *          password
 * @param ctx
 *          optional control object
 */
public void asyncOpenLedger(final long lId, final DigestType digestType, final byte[] passwd,
                            final OpenCallback cb, final Object ctx) {
    closeLock.readLock().lock();
    try {
        if (closed) {
            cb.openComplete(BKException.Code.ClientClosedException, null, ctx);
            return;
        }
        new LedgerOpenOp(BookKeeper.this, clientStats,
                         lId, digestType, passwd, cb, ctx).initiate();
    } finally {
        closeLock.readLock().unlock();
    }
}
```

Figure 5: Async Open Ledger pre miglioramenti

```java
 */
public void asyncOpenLedger(final long lId, final DigestType digestType, final byte[] passwd,
                            final OpenCallback cb, final Object ctx) {
    closeLock.readLock().lock();
    try {
        if (closed) {
            cb.openComplete(BKException.Code.ClientClosedException, null, ctx);
            return;
        }
        new LedgerOpenOp(BookKeeper.this, clientStats,
                         lId, digestType, passwd, cb, ctx).initiate();
    } finally {
        closeLock.readLock().unlock();
    }
}
```

Figure 6: Async Open Ledger post miglioramenti

```java
/**
 * Add entry to a ledger.
 */
public void addEntry(ByteBuf entry, boolean ackBeforeSync, WriteCallback cb, Object ctx, byte[] masterKey)
        throws IOException, BookieException, InterruptedException {
    long requestNanos = MathUtils.nowInNano();
    boolean success = false;
    int entrySize = 0;
    try {
        LedgerDescriptor handle = getLedgerForEntry(entry, masterKey);
        synchronized (handle) {
            if (handle.isFenced()) {
                throw BookieException
                        .create(BookieException.Code.LedgerFencedException);
            }
            entrySize = entry.readableBytes();
            addEntryInternal(handle, entry, ackBeforeSync, cb, ctx, masterKey);
        }
        success = true;
    } catch (NoWritableLedgerDirException e) {
        stateManager.transitionToReadOnlyMode();
        throw new IOException(e);
    } finally {
        long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
        if (success) {
            bookieStats.getAddEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
            bookieStats.getAddBytesStats().registerSuccessfulValue(entrySize);
        } else {
            bookieStats.getAddEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
            bookieStats.getAddBytesStats().registerFailedValue(entrySize);
        }

        entry.release();
    }
}
```

Figure 7: Add entry

```java
public ByteBuf readEntry(long ledgerId, long entryId)
        throws IOException, NoLedgerException {
    long requestNanos = MathUtils.nowInNano();
    boolean success = false;
    int entrySize = 0;
    try {
        LedgerDescriptor handle = handles.getReadOnlyHandle(ledgerId);
        if (LOG.isTraceEnabled()) {
            LOG.trace("Reading {}@{}", entryId, ledgerId);
        }
        ByteBuf entry = handle.readEntry(entryId);
        bookieStats.getReadBytes().add(entry.readableBytes());
        success = true;
        return entry;
    } finally {
        long elapsedNanos = MathUtils.elapsedNanos(requestNanos);
        if (success) {
            bookieStats.getReadEntryStats().registerSuccessfulEvent(elapsedNanos, TimeUnit.NANOSECONDS);
            bookieStats.getReadBytesStats().registerSuccessfulValue(entrySize);
        } else {
            bookieStats.getReadEntryStats().registerFailedEvent(elapsedNanos, TimeUnit.NANOSECONDS);
            bookieStats.getReadEntryStats().registerFailedValue(entrySize);
        }
    }
}
```

Figure 8: Read entry

```
        /**
         * Breaks a string into command + arguments.
         * @param cmdstring string of form "cmd arg1 arg2..etc"
         * @return true if parsing succeeded.
         */
        public boolean parseCommand(String cmdstring) {
            Matcher matcher = ARGS_PATTERN.matcher(cmdstring);

            List<String> args = new LinkedList<String>();
            while (matcher.find()) {
                String value = matcher.group(1);
                if (QUOTED_PATTERN.matcher(value).matches()) {
                    // Strip off the surrounding quotes
                    value = value.substring(1, value.length() - 1);
                }
                args.add(value);
            }
            if (args.isEmpty()) {
                return false;
            }
            command = args.get(0);
            cmdArgs = args;
            return true;
        }

    }
```

Figure 9: Parse Command

```java
/**
 * Parses a command line that may contain one or more flags
 * before an optional command string
 * @param args command line arguments
 * @return true if parsing succeeded, false otherwise.
 */
public boolean parseOptions(String[] args) {
    List<String> argList = Arrays.asList(args);
    Iterator<String> it = argList.iterator();

    while (it.hasNext()) {
        String opt = it.next();
        try {
            if (opt.equals("-server")) {
                options.put("server", it.next());
            } else if (opt.equals("-timeout")) {
                options.put("timeout", it.next());
            } else if (opt.equals("-r")) {
                options.put("readonly", "true");
            } else if (opt.equals("-client-configuration")) {
                options.put("client-configuration", it.next());
            }
        } catch (NoSuchElementException e) {
            System.err.println("Error: no argument found for option " + opt);
            return false;
        }

        if (!opt.startsWith("-")) {
            command = opt;
            cmdArgs = new ArrayList<String>();
            cmdArgs.add(command);
            while (it.hasNext()) {
                cmdArgs.add(it.next());
            }
            return true;
        }
    }
    return true;
}
```

Figure 10: Parse Options pre miglioramenti

```
public boolean parseOptions(String[] args) {
    List<String> argList = Arrays.asList(args);
    Iterator<String> it = argList.iterator();

    while (it.hasNext()) {
        String opt = it.next();
        try {
            if (opt.equals("-server")) {
                options.put("server", it.next());
            } else if (opt.equals("-timeout")) {
                options.put("timeout", it.next());
            } else if (opt.equals("-r")) {
                options.put("readonly", "true");
            } else if (opt.equals("-client-configuration")) {
                options.put("client-configuration", it.next());
            }
        } catch (NoSuchElementException e) {
            System.err.println("Error: no argument found for option " + opt);
            return false;
        }

        if (!opt.startsWith("-")) {
            command = opt;
            cmdArgs = new ArrayList<String>();
            cmdArgs.add(command);
            while (it.hasNext()) {
                cmdArgs.add(it.next());
            }
            return true;
        }
    }
    return true;
}
```

Figure 11: Parse Options post miglioramenti

```
public synchronized void cacheCommittedPacket(final QuorumPacket pkt) {
    committedPkts.add(pkt);
    pktsSize += LearnerHandler.packetSize(pkt);

    // remove 5 packets for every one added as we near the size limit
    for (int i = 0; pktsSize > pktsSizeLimit * 0.8 && i < 5; i++) {
        QuorumPacket oldPkt = committedPkts.poll();
        if (oldPkt == null) {
            pktsSize = 0;
            break;
        }
        pktsSize -= LearnerHandler.packetSize(oldPkt);
    }

    // enforce the size limit as a hard cap
    while (pktsSize > pktsSizeLimit) {
        QuorumPacket oldPkt = committedPkts.poll();
        if (oldPkt == null) {
            pktsSize = 0;
            break;
        }
        pktsSize -= LearnerHandler.packetSize(oldPkt);
    }
}
```

Figure 12: Cache committed packets

```
@Override
public synchronized long startForwarding(LearnerHandler learnerHandler, long lastSeenZxid) {
    Iterator<QuorumPacket> itr = committedPkts.iterator();
    if (itr.hasNext()) {
        QuorumPacket packet = itr.next();
        if (packet.getZxid() > lastSeenZxid + 1) {
            LOG.error(
                "LearnerHandler is too far behind (0x{} < 0x{}), disconnecting {} at {}",
                Long.toHexString(lastSeenZxid + 1),
                Long.toHexString(packet.getZxid()),
                learnerHandler.getSid(),
                learnerHandler.getRemoteAddress());
            learnerHandler.shutdown();
            return -1;
        } else if (packet.getZxid() == lastSeenZxid + 1) {
            learnerHandler.queuePacket(packet);
        }
        long queueHeadZxid = packet.getZxid();
        long queueBytesUsed = LearnerHandler.packetSize(packet);
        while (itr.hasNext()) {
            packet = itr.next();
            if (packet.getZxid() <= lastSeenZxid) {
                continue;
            }
            learnerHandler.queuePacket(packet);
            queueBytesUsed += LearnerHandler.packetSize(packet);
        }
        LOG.info(
            "finished syncing observer from retained commit queue: sid {}, "
                + "queue head 0x{}, queue tail 0x{}, sync position 0x{}, num packets used {}, "
                + "num bytes used {}",
            learnerHandler.getSid(),
            Long.toHexString(queueHeadZxid),
            Long.toHexString(packet.getZxid()),
            Long.toHexString(lastSeenZxid),
            packet.getZxid() - lastSeenZxid,
            queueBytesUsed);
    }
    activeObservers.add(learnerHandler);
    return lastProposedZxid;
}
```

Figure 13: Start Forwarding pre miglioramenti

```
@Override
public synchronized long startForwarding(LearnerHandler learnerHandler, long lastSeenZxid) {
    Iterator<QuorumPacket> itr = committedPkts.iterator();
    if (itr.hasNext()) {
        QuorumPacket packet = itr.next();
        if (packet.getZxid() > lastSeenZxid + 1) {
            LOG.error(
                "LearnerHandler is too far behind (0x{} < 0x{}), disconnecting {} at {}",
                Long.toHexString(lastSeenZxid + 1),
                Long.toHexString(packet.getZxid()),
                learnerHandler.getSid(),
                learnerHandler.getRemoteAddress());
            learnerHandler.shutdown();
            return -1;
        } else if (packet.getZxid() == lastSeenZxid + 1) {
            learnerHandler.queuePacket(packet);
        }
        long queueHeadZxid = packet.getZxid();
        long queueBytesUsed = LearnerHandler.packetSize(packet);
        while (itr.hasNext()) {
            packet = itr.next();
            if (packet.getZxid() <= lastSeenZxid) {
                continue;
            }
            learnerHandler.queuePacket(packet);
            queueBytesUsed += LearnerHandler.packetSize(packet);
        }
        LOG.info(
            "finished syncing observer from retained commit queue: sid {}, "
                + "queue head 0x{}, queue tail 0x{}, sync position 0x{}, num packets used {}, "
                + "num bytes used {}",
            learnerHandler.getSid(),
            Long.toHexString(queueHeadZxid),
            Long.toHexString(packet.getZxid()),
            Long.toHexString(lastSeenZxid),
            packet.getZxid() - lastSeenZxid,
            queueBytesUsed);
    }
    activeObservers.add(learnerHandler);
    return lastProposedZxid;
}
```

Figure 14: Start Forwarding post miglioramenti

```
public synchronized QuorumPacket removeProposedPacket(long zxid) {
    QuorumPacket pkt = proposedPkts.peek();
    if (pkt == null || pkt.getZxid() > zxid) {
        LOG.debug("ignore missing proposal packet for {}", Long.toHexString(zxid));
        return null;
    }
    if (pkt.getZxid() != zxid) {
        final String m = String.format(
            "Unexpected proposal packet on commit ack, expected zxid 0x%d got zxid 0x%d",
            zxid,  //bug
            pkt.getZxid());
        LOG.error(m);
        throw new RuntimeException(m);
    }
    proposedPkts.remove();
    return pkt;
}
```

Figure 15: Remove packet