

Report Software Testing

SOMMARIO

Sommario	1
1 Introduzione	2
2 JCS.....	2
2.1 JCSRemovalSimpleConcurrentTest.java	2
2.2 JCSThrashTest.java	3
2.3 Coverage.....	3
3 Bookkeeper.....	5
3.1 ReadCache	5
3.1.1 public void put (long ledgerId, long entryId, ByteBuf entry)	5
3.1.2 public ByteBuf get(long ledgerId, long entryId)	7
3.2 BufferedChannel.....	7
3.2.1 public void write(ByteBuf src)	7
3.2.2 public synchronized int read(ByteBuf dest, long pos, int length)	9
4 Syncope	12
4.1 GroupValidator	12
4.1.1 boolean isValid(final Group group, final ConstraintValidatorContext context)	12
4.2 RealmUtils.....	14
4.2.1 getGroupOwnerRealm(final String realmPath, final String groupKey).....	14
4.2.2 Optional<Pair<String, String>> parseGroupOwnerRealm(final String input).....	14
4.2.3 boolean normalizingAddTo(final Set<String> realms, final String newRealm)	15
4.2.4 Pair<Set<String>, Set<String>> normalize(final Collection<String> realms)	15
4.2.5 Set<String> getEffective(final Set<String> allowedRealms, final String requestedRealm)	16
5 Link.....	17
5.1 JCS.....	17
5.2 Bookkeeper.....	17
5.3 Syncope	17
6 Appendice.....	18

1 INTRODUZIONE

L'obiettivo di questo documento è quello di descrivere lo svolgimento delle attività di testing, motivando le scelte effettuate, e riportando i risultati ottenuti sui due progetti open-source analizzati (*Bookkeeper* e *Syncope*). A supporto di queste attività sono stati utilizzati diversi software e framework: Per l'implementazione dei casi di test sono stati utilizzati JUnit4 e Mockito, con Eclipse come IDE. Per la gestione del ciclo di build è stato utilizzato Maven, tramite il quale sono stati definiti degli appositi profili Jacoco per generare i report basati su control-flow coverage e Pit per eseguire il mutation testing. Le attività di building e testing sono state inserite in un contesto di Continuous Integration, in modo tale da analizzare i risultati ottenuti ad ogni commit e valutare se e come migliorare questi risultati. A tale scopo è stata utilizzata la piattaforma TravisCI, che ha permesso di buildare e testare il progetto in modo automatico su un ambiente controllato diverso da quello di sviluppo. Per avere inoltre un'analisi del codice ed un report sul coverage a seguito di ogni commit è stato utilizzato SonarCloud, integrandolo all'interno del ciclo di build maven.

2 JCS

JCS (Java Caching System) è un sistema di cache distribuito scritto in Java. Lo scopo di questa attività è quello di scrivere dei test parametrici, andando a convertire da JUnit3 a JUnit4 i test originali di JCS. Seguendo l'algoritmo di scelta, i test considerati sono `JCSRemovalSimpleConcurrentTest.java` e `JCSThrashTest.java`.

Per convertire i test da JUnit3 a JUnit4 si è cercato di mantenere inalterata l'implementazione dei test case, andando semplicemente ad annotare con `@Test` i metodi utilizzati nella Test Suite originale. Per rendere il test parametrico è stato quindi utilizzato il runner `Parametrized.class`, ed è stato definito un metodo `data()` con annotazione `@Parameters` per specificare i parametri da passare al costruttore della classe di test. Successivamente sono stati ricavati i parametri utilizzati nei vari casi di test, per poi andare ad inserirli nel metodo `data()` ed eseguire diversi test case in base ai diversi parametri considerati. Con un approccio di questo tipo risulta molto più semplice mantenere ed ampliare la test suite, in quanto per descrivere un nuovo caso di test basterà aggiungere una nuova entry nell'array ritornato dal metodo `data()`.

Per effettuare il setup dell'ambiente è stato definito il metodo `configure()` con annotazione `@BeforeClass`. Questo metodo, che viene eseguito una sola volta prima dell'esecuzione dei casi di test, si occupa di ottenere un'istanza di JCS e di impostare il corretto file `.ccf` di configurazione.

In linea generale per riscrivere i nuovi Test Case si è cercato di utilizzare un approccio di tipo black-box, basandosi sui commenti presenti nel codice e nei casi di test originali.

2.1 JCSREMOVALSIMPLECONCURRENTTEST.JAVA

L'obiettivo di questa classe è quello di testare che la pulizia della cache e la rimozione di entry vengano eseguite correttamente. In particolare si effettua il `put()` per inserire una o più entry in cache, e poi si esegue `remove()` o `clear()` per verificare che l'entry inserita sia stata rimossa correttamente.

Durante la progettazione del test parametrico è stato identificato come unico valore variabile l'intero `count`. Questo parametro definisce il numero di operazioni `put()` che verranno effettuate, e quindi il numero di entry che saranno prima inserite nella cache, e poi rimosse tramite `remove()` o `clear()`. Nei test originali di JCS il valore di `count` è sempre impostato a 500, quindi per una prima valutazione della copertura strutturale è stato utilizzato soltanto questo valore come input.

Altri possibili parametri utilizzati in JCS sono la coppia chiave/valore relativa all'entry che verrà inserita in cache. Tuttavia non è sembrato di particolare interesse far variare questi valori in quanto nel test originale viene semplicemente usata una stringa `"key"` ed una stringa `"value"` per ogni operazione di `put()`.

2.2 JCSTHRASHTEST.JAVA

L'obiettivo di questa classe è quello di testare l'inserimento di una entry, la rimozione, e la pulizia della cache, tramite i metodi `put()`, `remove()` e `clear()`. Inoltre viene anche valutato il comportamento della memoria dopo aver eseguito queste operazioni.

I valori individuati per la progettazione del test parametrico sono il numero di thread che vengono spawnati ed il numero di chiavi che vengono inserite nella cache. Questi valori nel test originale non sono definiti come variabili ma semplicemente come interi che vanno a definire l'end condition del ciclo for; di default vengono generati 15 thread ed inserite 500 chiavi. Sono state quindi rimossi questi valori hard-coded e al loro posto sono state inserite delle variabili di classe `numThreads` e `numKeys`. Per una prima valutazione sul coverage del test originale sono stati utilizzati soltanto i valori di default, ponendo quindi `numThreads=15` e `numKeys=500`.

Altre variabili utilizzate nel test originale sono le stringhe value/key che sono semplicemente inizializzate a "value" e "key". Tuttavia analizzando l'implementazione originale del test si è visto che non c'è nessuna asserzione o nessun controllo su eventuali eccezioni lanciate, per cui si è ritenuto non rilevante andare a parametrizzare questi valori testando ad esempio stringhe nulle o vuote. Infatti per ampliare i comportamenti catturati da questo test case sarebbe necessario modificarne l'implementazione, ma dovendo soltanto convertire il test si è ritenuto non conforme allo scopo di questa attività.

2.3 COVERAGE

Per analizzare il coverage ottenuto tramite le due classi di test è stato definito all'interno del `pom.xml` un profilo di coverage. Impostando questo profilo durante il build del progetto viene lanciato il plugin `jacoco:prepare-agent`. Non avendo il codice sorgente di JCS, *Jacoco* non può fornire nessun report sul coverage, per cui è necessario fornire al plugin una versione instrumented di `jcs.jar`. Al termine dell'esecuzione del goal `prepare-agent`, *Jacoco* genera un report di tipo `.exec`; al fine di analizzare tale report e valutare la copertura raggiunta, è necessario convertirlo in un formato human-readable. Per fare ciò è stato incluso nel profilo di coverage il plugin `exec-maven-plugin`, che si occupa di lanciare durante la fase di `maven:verify` uno script *bash* che converte il report `exec` in diversi formati, tra cui `html`.

Durante questa fase di analisi è stato utilizzato un approccio di tipo black-box, andando ad osservare il codice nello specifico in modo tale da motivare i risultati ottenuti dal report sul coverage. La classe stimolata dai due casi di test considerati è `org.apache.jcs.access.CacheAccess.java` ed in particolare vengono sollecitati i metodi `get()`, `put()`, `remove()` e `clear()`.

I risultati del coverage [Figura 1] mostrano che per i metodi `get()`, `put()` e `remove()` è stato raggiunto il 100% di coverage. Ciò vuol dire che i casi di test originali di JCS vanno già a testare tutte le istruzioni presenti nelle implementazioni di questi metodi.

Per quanto riguarda il metodo `clear()` è stata raggiunta invece una coverage del 45%. Analizzando il codice di questo metodo si è notato che il motivo per cui non si raggiunge il 100% di coverage è dovuto al fatto che l'implementazione dei Test JCS non prevede nemmeno un caso di test che mandi in errore la funzione `removeAll()` e che vada di conseguenza a stimolare il blocco catch lanciando l'eccezione `CacheException`.

```
public void clear() throws CacheException {
    try {
        this.cacheControl.removeAll();
    } catch ( IOException e ) {
        throw new CacheException( e );
    }
}
```

Per cercare di migliorare il coverage è stata quindi estesa la test suite; applicando *boundary value analysis* sono stati considerati come ulteriori valori anche 0, 1 e -1 per `count`, `numKeys` e `numThreads`. Tuttavia i risultati del nuovo report hanno mostrato una coverage per il metodo `clear()` ancora pari al 45%.

E' stato dunque effettuato un ulteriore tentativo per sviluppare un caso di test che vada a lanciare l'eccezione nel metodo `clear()`, andando a vedere direttamente l'implementazione del metodo `removeAll()`. Questo metodo invoca `CompositeCache.removeAll()`, ma anche analizzandone il codice non si è riusciti a trovare un possibile valore che permettesse di lanciare l'eccezione.

In conclusione, si è ritenuto che seguendo gli obiettivi ed i requisiti di questa attività, la copertura dei casi di test non fosse ulteriormente migliorabile, in quanto per migliorare il coverage del metodo `clear()` non basta ampliare la test suite, ma è necessario modificare e migliorare l'implementazione dei casi di test considerati.

3 BOOKKEEPER

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza ottimizzato per carichi di lavoro real-time. Le classi considerate per le attività di testing sono: `BufferedChannel` e `ReadCache`. La scelta di queste classi non si è basata sulle metriche calcolate durante l'analisi del codice, ma sulla chiarezza della documentazione, sulla quantità/qualità dei commenti, e sulla comprensibilità del ruolo e delle funzioni svolte dalla classe all'interno del progetto. I motivi per cui è stato utilizzato questo approccio sono principalmente due:

1. Nell'analisi del software effettuata, le metriche sono state calcolate soltanto per le release fino alla 4.5.0. Questo perché per il calcolo della buggyness e delle altre metriche, come da specifica, è necessario utilizzare i ticket di Jira ed effettuare un mapping con i relativi commit su GitHub. Tuttavia Bookkeeper dal 2017 ha cambiato l'ITS utilizzato, migrando da Jira a GitHub, per cui dopo la release 4.5.0 non si hanno più Ticket su Jira.
2. E' stato seguito durante le fasi di Domain Partitioning un approccio di tipo black-box, per poi migrare su un approccio white-box nel miglioramento della test-suite a seguito dell'analisi del coverage e del mutation testing. Per questo le classi scelte sono state quelle con la maggiore chiarezza delle specifiche, e con le quali si aveva una maggiore familiarità.

3.1 READCACHE

La classe `ReadCache` implementa una cache di lettura, ovvero uno spazio di memoria suddiviso in diversi segmenti, che viene utilizzata dal `LedgerStorage` per un più rapido accesso alle sue entry. Ogni entry viene identificata univocamente dall'id del ledger e dall'id dell'entry all'interno di quel ledger; prima di essere scritta in modo persistente sul log, l'entry viene inserita all'interno della cache. Relativamente a questa classe è stato valutato il comportamento dei metodi `put()` e `get()`, sviluppando quindi dei casi di test che andassero a stimolare principalmente questi due metodi.

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

Questo metodo copia nella cache il contenuto del `ByteBuf entry` passato come input. All'interno della cache l'entry viene identificata dal suo `entryId` e dal `ledgerId` in cui l'entry andrà scritta.

Applicando *Domain Partitioning* si è cercato innanzitutto di individuare i possibili valori assumibili dai parametri, basandosi sulla *semantica* del nome. I parametri `ledgerId` ed `entryId` sono degli *identificativi*, per cui ci si aspetta che debbano necessariamente essere non negativi. Per quanto riguarda invece il parametro `entry` non è stato possibile fare nessuna particolare assunzione, per cui si è considerato un'istanza valida, una non valida ed una nulla. La prima partizione in classi di equivalenza pertanto è data da:

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

Per lo sviluppo dei casi di test si è seguito un approccio di tipo *unidimensionale*, considerando quindi i vari parametri in modo indipendente e cercando di coprire con almeno un test tutte le classi di equivalenza definite in precedenza. Con entry *illegal* si intende un'entry con dimensione superiore alla dimensione della cache istanziata. Per la scelta dei valori di ogni classe di equivalenza è stata applicata *Boundary Values Analysis*, che ha portato allo sviluppo dei seguenti casi di test:

- **{ledgerId,entryId,entry}**
 - {1,0,valid_entry} → success
 - {0,1,illegal_entry} → memory exception
 - {1,-1,null_entry} → null pointer exception
 - {-1,1,valid_entry} → invalid argument exception
 - {0,-1,valid_entry} → invalid argument exception

Tutti i casi di test sviluppati hanno rispettato il comportamento atteso, ad eccezione del test case con `entryId` negativa `{0,-1,valid_entry}`. Dall'analisi preliminare ci si aspettava che `entryId` avesse un comportamento analogo al parametro `ledgerId`, e dunque che `entryId=-1` fosse un valore non valido; tuttavia questo caso di test non ha sollevato nessuna eccezione, e ciò ha quindi evidenziato come il parametro `entryId` accetti in realtà anche valori negativi. A seguito di questa osservazione è stato modificato il caso di test `{0,-1,valid_entry}`, settando a `null` l'eccezione attesa.

I risultati dei test hanno mostrato come per il metodo `put()` sia stata raggiunta una statement coverage pari al 76% ed una branch coverage pari al 75% [Figura 2]. Analizzando il coverage del metodo `put` fornito da Jacoco [Figura 3], si è visto che non vengono testate le istruzioni di riga 122 e 123 facenti riferimento ai segmenti e agli indici della cache. Per valutare anche questi statement si è deciso di stressare maggiormente la classe, permettendo di testare non soltanto l'inserimento di una singola entry, ma anche l'inserimento consecutivo di `N` entry.

Per mantenere inalterati i risultati ottenuti tramite *domain partitioning* è stato introdotto un nuovo parametro `doMultiplePut`, che quando impostato a `true` attiva un ciclo `for` ed effettua `N` `put` consecutive invece che un solo inserimento. Per tutti i test case sviluppati in precedenza il parametro `doMultiplePut` è stato impostato a `false`, mentre la test suite è stata estesa aggiungendo un caso di test che effettua `N` `put`; al termine, viene verificato che nella cache vi siano esattamente `N` entry e che la dimensione finale sia pari alla somma delle dimensioni delle entry inserite.

Il nuovo report sul coverage dopo il miglioramento della test suite riporta una statement coverage del 100% e branch coverage del 75% [Figura 4][Figura 5]. In particolare anche dopo aver ampliato la test suite non si è riusciti a coprire con i casi di test il branch di riga 113 `[offset+entrySize>segmentSize == false]`.

Si è cercato quindi di includere un nuovo caso di test in cui `offset + entrySize <= segmentSize`, ampliando la test suite includendo almeno un caso di test che vada a percorrere questo branch. Tuttavia, anche analizzando nello specifico l'implementazione del metodo `put`, non si è riusciti a sviluppare un tale test case, in quanto se viene soddisfatta la condizione `offset + entrySize <= segmentSize` si entra nel ramo `else` di riga 96, al termine del quale c'è il comando `return`. Di fatto quindi il branch `offset+entrySize<=segmentSize` di riga 113 sembrerebbe un branch non percorribile.

Al fine di migliorare l'adequatezza dei casi di test è stato successivamente applicato *Mutation Testing* tramite il framework *Pit*. I risultati [Figura 6] hanno mostrato come le mutazioni di riga 94 e 113 non siano state rilevate. Si è cercato quindi di migliorare l'implementazione del test program utilizzando il costruttore `ReadCache(allocator, maxCacheSize, maxSegmentSize)` che permette di specificare la massima dimensione assegnabile ai segmenti, anziché utilizzare quella di default come in precedenza. Sono stati inclusi quindi almeno un caso di test in cui `entrySize>maxSegmentSize>maxCacheSize`, uno in cui `entrySize<maxSegmentSize<maxCacheSize` ed uno in cui `entrySize>maxSegmentSize<maxCacheSize`. Tuttavia anche a seguito di tali modifiche sia la branch coverage che il mutation coverage sono rimasti invariati. Per migliorare i risultati ottenuti sarebbe quindi necessario uno studio approfondito del codice e del funzionamento riguardo il calcolo dell'offset e della dimensione del segmento. Tale analisi richiederebbe però un *effort* non banale, motivo per cui si è deciso di mantenere la test suite minimale, non includendo gli ultimi casi di test introdotti. In conclusione, la test suite è stata considerata come sufficientemente adeguata, avendo comunque raggiunto una statement coverage del 100% ed una branch coverage del 75%.

3.1.2 public ByteBuf get(long ledgerId, long entryId)

Questo metodo prende in input l'identificativo del ledger e l'identificativo dell'entry, ritornando in output il contenuto dell'entry se questa è presente in cache. Applicando *domain partitioning* si ha una prima partizione in classi di equivalenza, basata sul tipo di dato e sulla semantica del nome dei due parametri di input.

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

Dovendo testare il comportamento del metodo `get`, in fase di configurazione è stata inserita un'entry (`ledgerId, entryId`) = (1, 1) nella cache tramite `put`, e sono stati considerati degli ulteriori scenari per valutare la corretta lettura di tale entry dalla cache. In particolare è stato aggiunto un caso di test in cui la `get` ha successo (*hit*), ovvero l'entry cercata viene trovata in cache, ed un caso di test in cui la `get` non ha successo (*miss*), ovvero l'entry cercata non viene trovata nella cache. Applicando *boundary values* ed utilizzando un criterio di selezione *unidimensionale*, sono stati quindi definiti i seguenti test case:

- {ledgerId, entryId}
 - {1, 1} → hit
 - {2, 2} → miss
 - {1, -1} → invalid argument exception
 - {-1, 1} → invalid argument exception

Analogamente a quanto visto nell'analisi del metodo `put`, l'unico caso di test che non ha sperimentalmente rispettato il comportamento atteso è {1, -1} in cui `entryId < 0`; questo ha evidenziato come *bookkeeper* ammetta dei valori negativi come identificativo per le entry, almeno per quanto concerne la classe `ReadCache`. A seguito di questa osservazione è stato ridefinito il comportamento atteso dal test {1, -1}, che non dovrà sollevare nessuna eccezione, ma dovrà ritornare una entry nulla come risultato (miss in cache).

I primi risultati del coverage hanno mostrato come la test suite sia sufficientemente adeguata, poiché è stato raggiunto il 100% sia per lo statement coverage che per il branch coverage. [Figura 7][Figura 8].

Per un'ultima analisi è stato eseguito il *mutation testing*. Il report di *Pit* [Figura 9] ha mostrato come non tutti i mutanti vengano rilevati, ad esempio come nella mutazione di riga 151 in cui è stata rimossa la chiamata `lock.readLock().unlock()`. Si noti come questo sia un comportamento atteso in quanto anche senza togliere il lock di scrittura non va' in errore e non mostra nessun comportamento diverso rispetto a come è stato esplorato e testato il sistema. Un living mutant interessante è invece quello di riga 138 in cui viene sostituita la sottrazione con l'addizione; si è pensato dunque che la test suite non stimolasse adeguatamente il controllo sui segmenti, avendo effettuato il `put` e la `get` di una sola entry, pertanto è stato incluso un ulteriore caso di test in cui vengono effettuate 50 `get` consecutive di 50 entry differenti. Tuttavia anche a seguito di tale modifica il *mutation coverage* è restato inalterato; avendo dunque già raggiunto una coverage del 100%, non sono state effettuate ulteriori analisi, 0

la test suite è stata mantenuta in maniera minimale e considerata come sufficientemente adeguata.

3.2 BUFFEREDCHANNEL

Bookkeeper utilizza un file di log per il salvataggio delle entry. La classe `BufferedChannel` fornisce un livello di bufferizzazione davanti al `FileChannel` utilizzato per le operazioni di lettura e scrittura sul log. In questo modo si effettuano meno operazioni sul disco, velocizzando in generale la scrittura e lettura delle entries.

3.2.1 public void write(ByteBuf src)

Questo metodo scrive tutto il contenuto di `src` all'interno del buffer di scrittura, effettuando il flush su un file di log soltanto se la dimensione della cache non è sufficiente a contenere `src`. Applicando *domain partitioning* sono stati considerati come parametri dei test case l'input del metodo (`src`) e la capacità del `writeBuffer`, assegnata al momento della creazione del `BufferedChannel`. Non sono stati considerati invece i parametri `allocator` e `fc`, in quanto non ritenuti di interesse per stimolare e valutare il comportamento del metodo `write`; infatti andando ad impostare un `allocator` o un `fileChannel` nulli o non

validi semplicemente non viene permessa la creazione del `BufferedChannel`, e di conseguenza non è neanche possibile valutare il comportamento della scrittura. Una prima suddivisione in classi di equivalenza si è basata principalmente sul tipo di dato dei parametri, ed è data da:

- `src {valid,not_valid, null}`
- `capacity {<0,=0,>0}`

Analizzando il comportamento della classe `BufferedChannel` si è evidenziata sicuramente una correlazione tra l'input `src` ed il parametro `capacity` che specifica la dimensione del buffer di scrittura. In particolare il `BufferedChannel` tenta di scrivere l'entry sul `writeBuffer`; nel caso in cui il buffer non abbia spazio a sufficienza per contenere `src`, il `BufferedChannel` scrive direttamente sul `FileChannel` effettuando una scrittura persistente sul file di log per non perdere nessuna informazione.

A seguito di questa osservazione è stata pertanto modificata la partizione riguardante `src`, andando non più a considerare la validità o meno dell'entry, ma ponendola in relazione alla capacità assegnata al buffer di scrittura. Come parametro di test è stata quindi presa in considerazione la dimensione `srcSize` dell'entry, e non l'oggetto `src`. La nuova suddivisione in classi di equivalenza per i due parametri è stata definita come:

- `capacity {<0,=0,>0}`
- `src.size {<capacity,=capacity, >capacity}`

Applicando successivamente *boundary values analysis* sono stati ricavati i valori rappresentativi di ogni classe individuata:

- `capacity {-1,0,1,2}`
- `srcSize {capacity-1,capacity, capacity+1}`

Per quanto concerne il parametro `capacity` è stato considerato anche il valore 2 oltre al valore individuato come boundary; questo perché si è pensato che un buffer di capacità unitaria potesse non descrivere in modo abbastanza generale il comportamento del buffer. Seguendo un approccio unidimensionale sono stati quindi selezionati i seguenti casi di test:

- **{capacity,srcSize}**
 - o `{-1,=capacity} = {-1,-1}` → argument not valid exception
 - o `{0,capacity+1} = {0,1}` → src scritto su file
 - o `{1,capacity-1} = {1,0}` → nessuna scrittura
 - o `{2,capacity-1} = {2,1}` → nessun flush sul file
 - o `{2,capacity+1} = {2,3}` → flush sul file

Tutti i test case hanno rispettato a runtime il comportamento atteso, ad eccezione del test `{capacity,srcSize}={0,1}`. In questo caso infatti ci si aspettava una scrittura diretta sul file channel avendo un livello di buffering nullo. Tuttavia il programma entra in un ciclo infinito, senza mai terminare la propria esecuzione. Ciò ha evidenziato un probabile bug all'interno del codice in quanto `capacity=0` dovrebbe essere un valore non ammissibile, che però non viene gestito correttamente tramite il meccanismo delle eccezioni. Una rapida attività di debugging ha permesso di individuare la probabile causa dell'errore; essendo nulla la capacità del buffer, il programma cerca, correttamente, di effettuare immediatamente il flush di `src` sul `FileChannel`, ma l'end condition del while `copied<len` non viene mai soddisfatta. Il parametro `copied` viene infatti incrementato di una quantità intera `bytesToCopy`, valore che tuttavia risulta sempre nullo essendo calcolato come il minimo tra un certo valore ed il numero di byte scrivibili del `writeBuffer`. Avendo `BufferedChannel` capacità nulla, anche il relativo `writeBuffer` avrà capacità nulla, e di conseguenza `copied` sarà sempre pari a zero. Per far terminare l'esecuzione di questo test case, ed evidenziare il bug individuato, è stata utilizzata l'annotazione `@Test (timeout=500)`.

L'analisi del coverage [Figura 10] ha mostrato il raggiungimento del 74% di statement coverage e del 60% di branch coverage. Andando ad osservare il report [Figura 11] è stato evidenziato come non sia stato esplorato il branch `doRegularFlushes==true` (riga 141). La variabile booleana `doRegularFlushes` viene settata a `true` al momento in cui viene istanziato il `BufferedChannel`, soltanto se `unpersistedBytesBound` è maggiore di zero. Questo parametro rappresenta il *massimo numero di bytes che possono restare nel BufferedChannel senza essere stati scritti sul file*. Pertanto a livello pratico permette di effettuare la scrittura sul `FileChannel` non solo quando termina lo spazio nel buffer, ma al superamento di una certa soglia

prefissata. Al fine di migliorare l'adeguatezza della test suite è stata quindi migliorata l'implementazione del test program, istanziando il `BufferedChannel` tramite il costruttore `BufferedChannel(allocator, fc, capacity, unpersistedBytesBound)` impostando quindi anche il valore dell'`unpersistedBytesBound` considerato precedentemente sempre pari a zero. Sicuramente il programma avrà un diverso comportamento in base alla relazione che sussiste tra `unpersistedBytesBound` e `srcSize`, perciò le classi di equivalenza per tale parametro sono state definite come:

- `unpersistedBytesBound = {0, <srcSize, >srcSize}`

Per mantenere inalterati i risultati ottenuti in precedenza, in tutti i test case ricavati dal domain partitioning è stato esplicitato `unpersistedBytesBound=0`, mentre per estendere la test suite sono stati introdotti altri due casi di test; uno in cui `unpersistedBytesBound<srcSize` ed uno in cui `unpersistedBytesBound>srcSize`. Il caso `unpersistedBytesBound>capacity` non è stato preso in considerazione in quanto coincide sempre con il caso di `unpersistedBytesBound=0`. I valori per i nuovi test case non sono stati scelti tramite *boundary analysis* in quanto è già stato valutato il comportamento ai bordi del dominio e si è voluto inoltre testare il sistema anche per valori più grandi. La test suite è stata quindi ampliata aggiungendo i due seguenti casi di test:

- `{capacity, srcSize, unpersistedBytesBound}`
 - o `{6000, <capacity, <srcSize} = {6000, 4000, 3000}`
 - o `{6000, <capacity, >srcSize} = {6000, 4000, 5000}`

A seguito di tale miglioramento è stato raggiunto il 100% di statement coverage ed il 100% di branch coverage [Figura 12].

Per effettuare un'ultima valutazione sull'adeguatezza della test suite è stato applicato il *mutation testing*. Il report di *Pit* [Figura 13] ha mostrato come non siano state rilevate le mutazioni su `doRegularFlushes` (riga 136). Si noti come questo sia un comportamento atteso in quanto la mutazione introdotta è una *negated conditional* che disattiva il flush al superamento del bound specificato. Quindi come nei casi di test considerati inizialmente la scrittura sul log avverrà soltanto al superamento della capacità massima, e la mutazione non viene rilevata venendo in ogni caso scritta la stessa quantità di dati.

In conclusione non si è ritenuto necessario andare a migliorare ulteriormente la test suite, che è stata ritenuta adeguata avendo raggiunto il 100% sia di statement che di branch coverage.

3.2.2 `public synchronized int read(ByteBuf dest, long pos, int length)`

Il metodo `read` legge i dati dal `readBuffer` associato al `BufferedChannel` e li inserisce in un `ByteBuf` di destinazione passato come primo parametro. Inoltre il metodo accetta in input due interi `pos` e `length` che indicano rispettivamente la posizione da cui iniziare a leggere il buffer ed il numero di bytes da leggere a partire da `pos`.

Per istanziare il `BufferedChannel` bisogna passare come input il `FileChannel` associato al file che si vuole leggere e la capacità desiderata per i buffer di lettura e scrittura; quindi per configurare l'ambiente di test è stata creata una funzione `generateRandomFile` che genera un file temporaneo contenente dei bytes randomici. Per ogni caso di test sono stati dunque considerati, oltre ai parametri del metodo `read`, anche `capacity`, ovvero la dimensione del buffer di lettura, e `fileSize`, ovvero la dimensione del file che verrà generato. Anche in questo caso, come per il metodo `write`, non sono stati considerati `fileChannel` e `allocator` come parametri d'interesse per il test, in quanto un loro valore nullo semplicemente non permette di istanziare il `BufferedChannel`.

Applicando domain partitioning sono state inizialmente individuate le seguenti classi di equivalenza.

- `pos {<0, >=0}`
- `length {<0, >=0}`
- `fileSize {<0, >=0}`
- `capacity {<=0, >0}`

Analizzando il ruolo svolto da ognuno di questi parametri, è stata individuata una correlazione tra le variabili `length`, `pos` e `fileSize`; in particolare deve sicuramente valere la relazione per cui

`pos+length<=fileSize`, altrimenti si cercherebbe di effettuare una lettura oltre la fine del file. Pertanto le classi di equivalenza sono state raffinate in:

- `pos {<0,>=0}`
- `length {<0,>=0}`
- `fileSize {<0,<pos+length,>=pos+length}`
- `capacity {<=0,>0}`

Per la selezione dei valori è stata applicata inizialmente *boundary values analysis*, che è stata poi estesa aggiungendo i valori `length=2` e `capacity=2`. Questo perché si è ritenuto `length=1` non sufficientemente generale per testare il metodo in quanto va a leggere un solo byte. Analogamente `capacity=1` rappresenta un caso banale in cui i bytes vengono semplicemente letti e bufferizzati uno per volta. Le partizioni individuate sono quindi:

- `pos {-1,0,1}`
- `length {-1,0,1,2}`
- `fileSize {<0,pos+length-1,pos+length,pos+length}`
- `capacity {-1,0,1,2}`

Applicando selezione *unidimensionale* sono stati sviluppati i seguenti casi di test:

- **{fileSize,pos,length,capacity,srcSize}**
 - o `{-1,0,0,0}` → array exception
 - o `{0,-1,0,0}` → array exception
 - o `{0,0,-1,0}` → argument not valid exception
 - o `{0,0,0,-1}` → argument not valid exception
 - o `{0,0,0,0}` → nessuna lettura effettuata
 - o `{1,0,2,1}` → provo a leggere oltre la fine del file → `IOException`
 - o `{3,0,2,2}` → letti due bytes
 - o `{3,1,2,2}` → letto un byte

Il report di Jacoco [Figura 14] ha mostrato il raggiungimento di una statement coverage del 79% e branch coverage del 61%. Analizzando il report [Figura 15] si è visto come i casi di test non abbiano coperto le istruzioni di riga 264-266. Questo avviene perché nei test case sviluppati, il branch di riga 260 veniva esplorato soltanto nel caso `bytesToCopy==0` nel test case `{1,0,2,1}`. Pertanto veniva sempre lanciata l'eccezione `IOException` (riga 261) senza eseguire le successive istruzioni.

L'analisi ha dunque evidenziato come non siano state considerate delle funzionalità; il metodo `read` prima di leggere bytes dal `readBuffer` verifica se ci sono bytes ancora presenti nel `writeBuffer`, ma nella test suite non è stato incluso nessun caso di test che vada a stimolare questo comportamento. Per aumentare la copertura è stata innanzitutto migliorata l'implementazione del programma di test, permettendo in fase di configurazione dell'ambiente di poter scrivere un certo numero di bytes all'interno del `writeBuffer`. Successivamente è stata estesa la test suite prevedendo un ulteriore test case che vada a verificare che la lettura dal `writeBuffer` avvenga correttamente. Tuttavia anche con questo caso di test non sono state coperte le istruzioni 264-266, per cui è stato progettato un nuovo caso di test in cui si scrive nel `writeBuffer` in fase di configurazione, e si tenta successivamente di effettuare una lettura partendo da una posizione non concessa

- `{fileSize,pos,length,capacity,srcSize} = {5,9,3,2} → IOException`

A seguito di questa estensione lo statement coverage ha raggiunto il valore del 91% mentre il branch coverage è migliorato fino al 66% [Figura 16]. Si noti come questo rispecchia i risultati attesi in quanto ora vengono esplorati entrambi i branch di riga 260 e di conseguenza vengono anche coperte le istruzioni 264-266 [Figura 17].

Per valutare e migliorare ulteriormente l'adeguatezza della test suite è stato applicato *mutation testing*. Il report di Pit [Figura 18] ha mostrato come diversi mutanti non siano stati rilevati. Si noti come non avendo raggiunto una statement coverage del 100% è naturale che i mutanti relativi a branch non coperte non vengano rilevati (righe 251/262/266).

Un mutante non-killed interessante mostrato dal report è quello di riga 278 in cui non è stato rilevato un diverso comportamento cambiando la boundary di `if (readBytes <= 0)`. Questo ha evidenziato come probabilmente non abbiamo esplorato tutti i possibili valori di ritorno della funzione `FileChannel.read()`. Leggendo la documentazione il metodo `read()` di `FileChannel` ritorna un valore negativo se la posizione

fornita è maggiore o uguale alla dimensione del file. Un primo caso di test aggiunto è stato quello in cui semplicemente `pos > fileSize`. Tuttavia il metodo `BufferedChannel.read()` valida questo tipo di input prima ancora di effettuare la lettura sul `FileChannel`, sollevando un'eccezione prima ancora di leggere dal `FileChannel`. Dunque, al fine di triggerare `readBytes == -1` è stato incluso un caso di test con `capacity=0` e `fileSize, pos, length` diversi da zero.

- `{fileSize,pos,length,capacity,srcSize} = {5,1,4,0} → IOException`

Il nuovo report Pit [\[Figura 19\]](#) ha mostrato come la mutazione venga ora rilevata. Anche il report Jacoco ha mostrato dei miglioramenti sul coverage, avendo raggiunto una statement coverage del 95% e branch coverage del 72%. [\[Figura 20\]](#) [\[Figura 21\]](#)

Andando ad analizzare le motivazioni per cui si è raggiunta una statement coverage soltanto del 72%, è stato evidenziato come questa sia dovuta alle branch non esplorate in cui `writeBuffer == null`. Nel costruttore utilizzato per istanziare il `BufferedChannel`, con `capacity` si indica la dimensione sia del `readBuffer` che del `writeBuffer`, e non è stato quindi mai utilizzato un `writeBuffer` nullo. Per cercare di esplorare questo branch è stata quindi modificata l'implementazione del test program, istanziando il `BufferedChannel` tramite il costruttore:

```
BufferedChannel(ByteBufAllocator allocator, FileChannel fc, int writeCapacity, int readCapacity, ...)
```

Tuttavia ponendo `writeCapacity=0` viene istanziato un `writeBuffer` di dimensione zero, ma non nullo, mentre con `writeCapacity=null` viene lanciata una `NullPointerException`. Pertanto lo scenario con `writeCapacity==null` sembra impossibile da proporre dei nostri casi di test in quanto il buffer di scrittura viene istanziato automaticamente al momento della creazione del `BufferedChannel`.

In conclusione, anche in virtù di quest'ultima osservazione, si è ritenuto che la test suite prodotta sia sufficientemente adeguata. La branch coverage del 72% è relativamente bassa in quanto vengono esplorati 13 branch su 18 totali. Tuttavia, come osservato in precedenza, la condizione `writeBuffer == null` non è mai soddisfacibile, per cui 4 branch (*righe 251/262*) non sono in realtà percorribili. Volendo normalizzare la copertura raggiunta, si può considerare come siano stati raggiunti 13 branch su 14 totali percorribili, con una branch coverage 'reale' del 93%.

Per quanto concerne la statement coverage, l'unica istruzione non coperta è il `break` di linea 264, che non viene mai eseguita in quanto non viene esplorato il relativo branch con `writeBuffer` nullo. Di fatto, questo statement è quindi considerabile come *dead code*, e di conseguenza la statement coverage 'reale' può essere considerata pari al 100%.

4 SYNCOPÉ

Apache Syncope è un sistema Open Source per la gestione delle identità digitali in ambienti aziendali. La gestione delle identità (*IdM*) consiste nel gestire i dati degli utenti su sistemi e applicazioni, ed implica la gestione degli attributi, dei ruoli, delle risorse e dei diritti degli utenti. L'obiettivo è quindi quello di definire con chiarezza chi ha accesso a *cosa*, *come*, *quando* e *perché*. Le classi considerate per le attività di testing sono: `GroupValidator` e `RealmUtils`. La scelta di queste classi non si è basata sulle metriche ricavate durante l'analisi del codice, bensì sulla chiarezza della documentazione e del ruolo svolto all'interno del progetto. Il motivo principale per cui è stato utilizzato questo criterio di selezione è dovuto al fatto che si è seguito un approccio di tipo black-box durante la prima fase di domain partitioning e boundary analysis, per poi migrare su un approccio più di tipo white-box nel miglioramento della test suite a seguito dei report Jacoco e Pit. Per questo le classi scelte sono state quelle con la maggiore chiarezza delle specifiche, e con le quali si aveva una maggiore conoscenza/familiarità.

4.1 GROUPVALIDATOR

Questa classe si occupa di validare un Gruppo, ovvero verificare che l'oggetto `Group` rispetti le regole ed i template definiti dalla specifica. I gruppi in Syncope hanno un duplice scopo: definire un insieme di `User` o `AnyObject` per implementare il group-based provisioning, ad esempio per associare in maniera dinamica i membri a delle risorse esterne, e rappresentare questo insieme di entità su risorse esterne che supportano il concetto di gruppo. La classe `GroupValidator` quindi in particolare controlla che:

- Il nome del gruppo sia valido.
- Il gruppo abbia una ownership valida.
- I membri del gruppo abbiano una membership valida.

4.1.1 boolean isValid(final Group group, final ConstraintValidatorContext context)

La classe `GroupValidator` utilizza un unico metodo `isValid` per validare il gruppo considerato. Per sviluppare i casi di test e valutare il comportamento di tale classe è stato utilizzato il framework Mockito, in modo da potersi focalizzare sul testing del validatore senza preoccuparsi di una reale e corretta istanziazione del gruppo da validare. A tale scopo è stata definita una stringa `groupType` come parametro di test; in base al valore di questo parametro (`NULL`, `VALID`, `NOT_VALID`) si effettua uno specifico setup della mocked instance del gruppo, descrivendone un diverso comportamento tramite le istruzioni `when` e `thenReturn`.

Il metodo `isValid` prende in input il gruppo da validare ed un contesto. Anche per quanto riguarda il contesto è stato utilizzato Mockito in modo tale da simulare il comportamento dei diversi contesti senza istanziare realmente un `ConstraintValidatorContext`. Analogamente a quanto fatto per il gruppo, è stato considerato un parametro di test `contextType` ed in base al valore di questo parametro è stato effettuato un diverso setup dello stub relativo al contesto.

Applicando domain partitioning sono state ricavate le seguenti classi di equivalenza per i parametri `groupType` e `contextType`:

- `groupType` = {`valid`, `not_valid`, `null`}
- `contextType` = {`valid`, `null`}

Per `groupType=not_valid` si è considerato un gruppo con nome nullo. Per `contextType` invece il caso `null` è stato considerato equivalente al caso `not_valid` in quanto in entrambi i casi non è possibile impostare una `ConstraintViolation`. I casi di test sono stati selezionati applicando un criterio unidimensionale:

- **{groupType, contextType}**
 - {`valid`, `valid`} → `isValid == true`
 - {`valid`, `null`} → `NullPointerException`
 - {`not_valid`, `valid`} → `isValid == false`
 - {`null`, `valid`} → `NullPointerException`

Tutti i casi di test considerati hanno rispettato il comportamento previsto, ed il report di Jacoco [Figura 22] ha mostrato una statement coverage ed una branch coverage soltanto del 50%. Analizzando il report [Figura 23] si è visto come siano stati esplorati soltanto 1/4 dei branch di *riga 38*, 2/6 dei branch di *riga 47*, 1/2 dei branch di *riga 57* e 2/4 dei branch di *riga 70*. Pertanto sono stati esaminati nello specifico ognuno dei *missed branch* evidenziati, al fine di migliorare la copertura strutturale della test suite.

Il *branch di riga 38* controlla quali sono i proprietari del gruppo; nel domain partitioning è stato considerato come `VALID_GROUP` soltanto un gruppo senza nessun proprietario, in cui non è stato impostato né un `groupOwner` né un `userOwner`. Pertanto sono state ampliate le tipologie di gruppo considerate, inserendo un gruppo con proprietario un utente (`USER_OWNED_GROUP`), un gruppo con proprietario un altro gruppo (`GROUP_OWNED_GROUP`) ed un gruppo con proprietario sia un utente che un gruppo (`BOTH_OWNED_GROUP`). I casi di test aggiunti sono quindi:

- **{groupType,contextType}**
 - o {`USER_OWNED_GROUP`,`valid`} → `isValid == true`
 - o {`GROUP_OWNED_GROUP`,`valid`} → `isValid == true`
 - o {`BOTH_OWNED_GROUP`,`valid`} → `isValid == false`

Il *branch di riga 47* riguarda la validazione del nome del gruppo; nel domain partitioning è stato considerato come `NOT_VALID_GROUP` soltanto un gruppo con nome nullo, ma in realtà la validazione consiste anche nel verificare che il nome rispetti un pattern specifico (`SyncopConstants.NAME_PATTERN`).

Approfondendo la struttura di tale pattern si è visto come nel nome siano ammessi soltanto caratteri alfanumerici ed i caratteri speciali `{-@.~}+`. `NOT_VALID_GROUP` è stato quindi specializzato in due diverse categorie; `NOT_VALID_GROUP_NULL` rappresenta un gruppo con nome nullo (già considerato), mentre `NOT_VALID_GROUP_CHAR` rappresenta un gruppo che non rispetta il pattern richiesto. In questo caso quindi è stato inserito nel nome il carattere non valido `'&'`. Di conseguenza la test suite è stata ampliata aggiungendo il seguente caso di test:

- **{groupType,contextType}**
 - o {`NOT_VALID_GROUP_CHAR`,`valid`} → `isValid == false`

Il *branch di riga 57* controlla che per ogni membro del gruppo sia impostata la corretta membership dinamica. In particolare non sono permesse membership di tipo `Group` o `User`. Nel domain partitioning non era stata impostata la membership per i membri del gruppo, quindi è stato raffinato il setup della mocked instance definendo tramite `thenReturn` il comportamento del metodo `getAdynMembership()`. Successivamente sono state specializzate ulteriormente le tipologie di gruppo considerate in modo da coprire questa funzionalità nei casi di test. Per quanto riguarda `VALID_GROUP` è stata aggiunta una membership conforme a `AnyTypeKind.ANY_OBJECT`. Per quanto riguarda invece la casistica dei gruppi non validi è stata aggiunta una nuova categoria `NOT_VALID_GROUP_MEMB` in cui la membership è conforme a `AnyTypeKind.USER` invece che a `AnyTypeKind.ANY_OBJECT` come richiesto dalla specifica. E' stato quindi aggiunto il seguente caso di test:

- **{groupType,contextType}**
 - o {`NOT_VALID_GROUP_MEMB`,`valid`} → `isValid == false`

Nel *branch di riga 70* viene effettuato un ulteriore controllo sulla membership dei membri del gruppo. In particolare i diversi membri del gruppo non possono avere lo stesso tipo di `DynMembership`. E' stata quindi aggiunta un'altra tipologia di gruppo, avente due membri ed entrambi con la stessa `DynMembership`. In questo modo il gruppo avrà dimensione 2, mentre l'array contenente le diverse tipologie di membership avrà dimensione 1. Aggiungendo quindi il seguente caso di test si è riusciti ad esplorare anche il *missed branch* `[isValid&&anyTypes.size()<group.getADynMemberships().size()]`.

- **{groupType,contextType}**
 - o {`NOT_VALID_GROUP_MEMB_DIFF`,`valid`} → `isValid == false`

Il nuovo report di jacoco mostra come sia stato raggiunto il 100% sia per statement coverage che per branch coverage [Figura 24]. Anche applicando mutation testing, il report di Pit mostra che sono state rilevate tutte le mutazioni [Figura 25]. Possiamo quindi concludere che la test suite sviluppata per la classe `GroupValidator` sia sufficientemente adeguata.

4.2 REALMUTILS

Questa classe fornisce dei metodi di supporto ai realms. Un `Realm` definisce un albero gerarchico nel dominio della sicurezza, utilizzato principalmente per contenere Groups, Users e AnyObjects.

4.2.1 `getGroupOwnerRealm(final String realmPath, final String groupKey)`

Questo metodo prende in input il path del realm e la chiave identificativa del gruppo, fornendo in output una stringa nel formato `path@key` utilizzato da Syncopé per descrivere un `GroupOwner`. Essendo entrambi i parametri di tipo stringa sono state considerate come partizioni `{valid, empty, null}` ed utilizzando selezione unidimensionale sono stati individuati i seguenti casi di test:

- `{realmPath, groupKey}`
 - o `{valid, valid}`
 - o `{valid, empty}` → `Illegal Argument Exception`
 - o `{empty, valid}` → `Illegal Argument Exception`
 - o `{valid, null}` → `Null Pointer Exception`
 - o `{null, valid}` → `Null Pointer Exception`

Contrariamente a quanto atteso, i casi di test con `realmPath` e `groupKey` nulli o vuoti non hanno lanciato nessuna eccezione. Analizzando l'implementazione del metodo `getGroupOwnerRealm`, si è visto come questo comportamento non sia in realtà anomalo; prima di generare la stringa `realmPath@groupKey` non viene effettuata nessuna validazione dell'input, per cui sono ammesse anche stringhe vuote. Inoltre non viene utilizzata la funzione `String.concat()` ma semplicemente l'operatore `+`, il quale non dà nessun errore anche con parametri nulli. E' stata quindi minimizzata la test suite, mantenendo come unico caso di test `{realmPath, groupKey}={valid, valid}`, in quanto presenta lo stesso comportamento di tutti i casi di test considerati dal *domain partitioning*.

Anche con un unico caso di test, la test suite risulta sufficientemente adeguata avendo raggiunto il 100% di statement coverage [Figura 26][Figura 27]. Ad ulteriore conferma della bontà di tale test suite, anche applicando mutation testing tutte le mutazioni generate sono state killate [Figura 28].

4.2.2 `Optional<Pair<String, String>> parseGroupOwnerRealm(final String input)`

Questo metodo offre la funzionalità opposta al metodo `getGroupOwnerRealm`; data una stringa in input nel formato `path@key`, il metodo fornisce in output una coppia di valori `<path, key>`. L'unico parametro di input è stato quindi diviso in 3 classi di equivalenza `{valid, not_valid, null}` e per ognuna di queste classi è stato considerato un caso di test.

- `{input}`
 - o `valid`: stringa correttamente formattata come `path@key`. La stringa verrà splittata per cui ci si attende che la coppia contenga correttamente `<path, key>`.
 - o `not_valid`: stringa formattata erroneamente come `path_key`. La stringa non verrà splittata per cui ci si attende che la coppia ritornata abbia il campo `key` vuoto.
 - o `null`: stringa nulla. Ci si attende che il metodo `split` lanci una `NullPointerException`.

I casi di test individuati rispettano tutti il comportamento atteso. Il report sul coverage ha mostrato come sia stata raggiunta una statement coverage del 100% ed una branch coverage del 75% [Figura 29]. Analizzando il report [Figura 30] si è visto come non sia stato esplorato nella test suite il branch di riga 38 dove `split==null`. In realtà, andando a leggere la documentazione del metodo `String.split()` è stato osservato come questo non possa mai ritornare un valore `null`, in quanto anche passando un parametro nullo o una stringa vuota fornisce in output un array vuoto `≠ null`. La branch coverage del 75% fa riferimento al numero totale di branch nel codice e non rispetto ai branch raggiungibili; essendo tuttavia `split==null` una condizione non soddisfacibile, la test suite è stata considerata sufficientemente adeguata e non ulteriormente migliorabile. Ad ulteriore conferma sull'adeguatezza, applicando mutation testing tutte le mutazioni vengono correttamente individuate [Figura 31].

4.2.3 boolean normalizingAddTo(final Set<String> realms, final String newRealm)

Questo metodo prende in input un insieme di realms, ed un nuovo realm che deve essere aggiunto a quell'insieme; i realms sono rappresentati tramite stringhe. In output il metodo fornisce un valore booleano che indica se l'inserimento del realm è stato eseguito oppure no. Applicando domain partitioning una partizione individuata per i due parametri è data da:

- `realms = {all_valid, all_not_valid, empty_set}`
- `newRealm = {valid, not_valid, empty}`

Per definire i valori di un realm valido (che verrà aggiunto) o non valido (che non verrà aggiunto) è stato necessario studiare più da vicino l'implementazione del metodo, non essendo presente alcuna documentazione a riguardo. In particolare si è visto che per l'aggiunta di `newRealm` al set possono presentarsi tre diversi scenari:

1. Se `newRealm` è contenuto all'interno di almeno uno dei realm presenti nel set `realms`, allora viene ritornato `false`, e `newRealm` non verrà aggiunto all'insieme.
2. Per ogni realm `r` all'interno del set, se `r` è contenuto all'interno di `newRealm` allora viene ritornato `true`, `r` viene rimosso da `realms`, e al suo posto viene inserito `newRealm`.
3. Nessuna delle due precedenti, `newRealm` verrà aggiunto al set `realm`.

A seguito di queste informazioni è stata evidenziata dunque una correlazione tra `newRealm` e l'insieme `realms`. In particolare `newRealm` non è a prescindere valido o non valido, ma lo è soltanto rispetto all'insieme fornito in input; analogamente, `realms` non è a prescindere un insieme valido o non valido, ma dipende se qualche suo elemento è contenuto all'interno di `newRealm`. Pertanto i casi di test non hanno seguito direttamente il *domain partitioning*, ma si è ritenuto più accurato svilupparli a partire dalle precedenti osservazioni, andando ad includere almeno un caso di test in cui:

- `newRealm` e `realms` sono `VALID`, nel senso che `newRealm` non inizia con nessun realm appartenente al set, e viceversa. Il comportamento atteso è che il metodo ritorni `true` e che l'entry venga aggiunta al set senza nessuna rimozione (caso 3).
- `newRealm` è `CONTAINED` nel set `realms`, nel senso che `newRealm` inizia per almeno uno dei realm appartenenti al set. Ci si aspetta che il metodo torni `false` e che `newEntry` non venga aggiunta al set (caso 1).
- `realms` è `CONTAINED` in `newRealm`, nel senso che almeno un elemento del set `realms` inizia con la stringa `newRealm`. Ci si aspetta che il metodo torni `true`, che gli elementi contained vengano rimossi dal set, e che `newRealm` venga aggiunta al set (caso2).

A livello implementativo i casi di test hanno come parametro un'unica stringa `realmsContext` che specifica quale dei tre scenari visti in precedenza si vuole andare a testare; in base alla stringa passata verrà effettuato un differente setup dei due parametri `realms` e `newRealms`. Per verificare il risultato atteso sono state utilizzate due asserzioni: una sul booleano ritornato dal metodo ed una sulla dimensione finale di `realms`. Pertanto in base al differente `realmsContext` considerato saranno anche presenti delle differenti asserzioni.

Tutti i casi di test hanno rispettato il comportamento atteso, ed è stata raggiunta una statement coverage e branch coverage del 100% [Figura 32] [Figura 33].

Per un'ultima valutazione sull'adeguatezza della test suite prodotta è stato eseguito il framework Pit per il mutation testing; tutte le mutazioni prodotte vengono killate, e si è ritenuta pertanto la test suite come sufficientemente adeguata [Figura 34].

4.2.4 Pair<Set<String>, Set<String>> normalize(final Collection<String> realms)

Questo metodo prende in input una collezione di Realm e la *normalizza*; in particolare vengono rimossi i realms non validi usando in modo iterativo `normalizingAddTo()`, e vengono estratti eventuali *groupOwner* contenuti nella collezione, se espressi correttamente nella forma `path@key`. L'insieme dei realms normalizzati e l'insieme di tutti i group owner trovati vengono forniti in output tramite un `Pair` dei due insiemi.

L'unico parametro di questo metodo è `Collection<String> realms`, che in una prima fase di domain partitioning è stato partizionato come `realms = {all_valid, all_not_valid, null}`. Avendo già testato le funzionalità del metodo `normalizingAddTo()`, non si è ritenuto di particolare interesse valutare il comportamento della normalizzazione, pertanto non sono stati considerati le varie tipologie di realms che rientrassero in tutte le casistiche viste nel test precedente. Le partizioni sono state quindi definite considerando la collezione `realms` come `valid/not_valid` in base alla presenza o meno dei `groupOwner` al suo interno. La classe `with_group_owner` rappresenta una collezione di realms in cui almeno un realm è nella forma `path@key`, mentre `without_group_owner` rappresenta una collezione di realms in cui nessuno dei realm è nella forma `path@key`. I casi di test sono stati scelti in modo minimale, includendo un solo test per ogni casistica:

- `Realms=with_group_owner`, atteso `set groupOwnership` non vuoto.
- `Realms=without_group_owner`, atteso `set groupOwnership` vuoto.
- `Realms=null`, ci si aspetta che gli insiemi `normalized` e `groupOwnership` siano entrambi vuoti.

Tutti i casi di test hanno rispettato il comportamento atteso. Il report Jacoco ha mostrato una statement coverage ed una branch coverage del 100% [Figura 35] [Figura 36]. Anche applicando mutation testing tutte non rimangono living-mutants per cui si è ritenuto che la test suite prodotta sia sufficientemente adeguata [Figura 37].

4.2.5 `Set<String> getEffective(final Set<String> allowedRealms, final String requestedRealm)`

Questo metodo prende in input un insieme `allowedRealms` ed un `requestedRealm`. In output viene fornito un insieme di realms filtrato e normalizzato. Applicando *domain partitioning* e basandosi sul tipo di dato sono state trovate le seguenti partizioni per i due parametri di input.

- `allowedRealms {null, all_valid}`
- `requestedRealm {null, valid}`

Applicando una selezione unidimensionale sono stati individuati i seguenti casi di test:

- **`{allowedRealms, requestedRealm}`**
 - o `{all_valid, valid}` → `allowedRealm == effective`
 - o `{valid, null}` → `nullPointer`
 - o `{null, valid}` → `nullPointer`

Il test case `{null, valid}` non ha rispettato il comportamento previsto, in quanto con `allowedRealms null` non viene lanciata nessuna eccezione e viene ritornato un insieme vuoto. Analizzando l'implementazione del metodo `getEffective` si è visto come questo comportamento sia legittimo, in quanto la funzione `normalize` anche a partire da un insieme di input nullo, fornisce in output un insieme vuoto. Pertanto è stata modificata l'implementazione del test program, ponendo come risultato atteso un insieme `effective` vuoto anziché il lancio dell'eccezione.

Il report sul coverage ha mostrato come sia stata raggiunto il 100% di branch coverage e di statement coverage [Figura 38] [Figura 39]. Applicando mutation testing viene prodotta una sola mutazione, che viene tuttavia individuata correttamente dalla nostra test suite [Figura 40]. Pertanto i casi di test sviluppati sono stati considerati come sufficientemente adeguati anche per il metodo `getEffective`.

5 LINK

5.1 JCS*

- Repository GitHub: <https://github.com/danilo-dellorco/jcsTests>
- Travis CI: <https://travis-ci.com/github/danilo-dellorco/jcsTests>

5.2 BOOKKEEPER

- Repository GitHub: <https://github.com/danilo-dellorco/bookkeeper>
- Travis CI: <https://travis-ci.com/github/danilo-dellorco/bookkeeper>
- Sonarcloud: https://sonarcloud.io/dashboard?id=danilo-dellorco_bookkeeper

5.3 SYSCOPE

- Repository GitHub: <https://github.com/danilo-dellorco/syncope>
- Travis CI: <https://travis-ci.com/github/danilo-dellorco/syncope>
- Sonarcloud: https://sonarcloud.io/dashboard?id=danilo-dellorco_syncope

**Per l'attività di JCS non è stato utilizzato SonarCloud in quanto l'analisi del codice è stata effettuata esclusivamente in locale non avendo a disposizione il codice sorgente del progetto*

6 APPENDICE

Figura 1 – Coverage JCS





























































Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• get(Object)		100%		100%	0	2	0	2	0	1
• put(Object, Object)		100%		n/a	0	1	0	2	0	1
• remove(Object)		100%		n/a	0	1	0	2	0	1
• CacheAccess(CompositeCache)		100%		n/a	0	1	0	3	0	1
• getStatistics()		100%		n/a	0	1	0	1	0	1
• getStats()		100%		n/a	0	1	0	1	0	1
• dispose()		100%		n/a	0	1	0	2	0	1
• static {...}		90%		50%	1	2	0	1	0	1
• put(Object, Object, IElementAttributes)		60%		50%	2	3	4	11	0	1
• clear()		45%		n/a	0	1	2	5	0	1
• freeMemoryElements(int)		0%		n/a	1	1	8	8	1	1
• resetElementAttributes(Object, IElementAttributes)		0%		0%	2	2	5	5	1	1
• putSafe(Object, Object)		0%		0%	2	2	4	4	1	1
• ensureCacheManager()		0%		0%	3	3	5	5	1	1
• getElementAttributes(Object)		0%		n/a	1	1	6	6	1	1
• destroy()		0%		n/a	1	1	5	5	1	1
• defineRegion(String, ICompositeCacheAttributes, IElementAttributes)		0%		n/a	1	1	2	2	1	1
• defineRegion(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
• getAccess(String, ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
• defineRegion(String)		0%		n/a	1	1	2	2	1	1
• getAccess(String)		0%		n/a	1	1	2	2	1	1
• destroy(Object)		0%		n/a	1	1	2	2	1	1
• getCacheElement(Object)		0%		n/a	1	1	1	1	1	1
• resetElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
• setDefaultElementAttributes(IElementAttributes)		0%		n/a	1	1	2	2	1	1
• setCacheAttributes(ICompositeCacheAttributes)		0%		n/a	1	1	2	2	1	1
• getElementAttributes()		0%		n/a	1	1	1	1	1	1
• getDefaultElementAttributes()		0%		n/a	1	1	1	1	1	1
• getCacheAttributes()		0%		n/a	1	1	1	1	1	1
• remove()		0%		n/a	1	1	2	2	1	1
Total		242 of 327		11 of 16		31%		27 38		63 87 20 30

Figura 2 – Coverage put











Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• put(long, long, ByteBuf)		76%		75%	1	3	3	21	0	1
• size()		87%		83%	1	4	1	9	0	1
• close()		100%		n/a	0	1	0	2	0	1
• ReadCache(ByteBufAllocator, long)		100%		n/a	0	1	0	2	0	1
• count()		100%		100%	0	2	0	6	0	1

Figura 3 – Analisi coverage del metodo put

```

86. public void put(long ledgerId, long entryId, ByteBuf entry) {
87.     int entrySize = entry.readableBytes();
88.     int alignedSize = align64(entrySize);
89.
90.     lock.readLock().lock();
91.
92.     try {
93.         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94.         ◆ if (offset + entrySize > segmentSize) {
95.             // Roll-over the segment (outside the read-lock)
96.         } else {
97.             // Copy entry into read cache segment
98.             cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99.                 entry.readableBytes());
100.             cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101.             return;
102.         }
103.     } finally {
104.         lock.readLock().unlock();
105.     }
106.
107.     // We could not insert in segment, we to get the write lock and roll-over to
108.     // next segment
109.     lock.writeLock().lock();
110.
111.     try {
112.         int offset = currentSegmentOffset.getAndAdd(entrySize);
113.         ◆ if (offset + entrySize > segmentSize) {
114.             // Rollover to next segment
115.             currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116.             currentSegmentOffset.set(alignedSize);
117.             cacheIndexes.get(currentSegmentIdx).clear();
118.             offset = 0;
119.         }
120.
121.         // Copy entry into read cache segment
122.         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(), entry.readableBytes());
123.         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
124.     } finally {
125.         lock.writeLock().unlock();
126.     }
127. }

```

Figura 4 – Report coverage put (dopo il miglioramento della test suite)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● put(long, long, ByteBuf)	<div><div></div></div>	100%	<div><div></div></div>	75%	1	3	0	22	0	1
● ReadCache(ByteBufAllocator, long, int)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	12	0	1
● size()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	4	0	9	0	1
● count()	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	6	0	1
● ReadCache(ByteBufAllocator, long)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	2	0	1
● close()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	2	0	1

Figura 5 - Analisi coverage del metodo put (dopo miglioramento della test suite)

```

86. public void put(long ledgerId, long entryId, ByteBuf entry) {
87.     int entrySize = entry.readableBytes();
88.     int alignedSize = align64(entrySize);
89.
90.     lock.readLock().lock();
91.
92.     try {
93.         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94.         if (offset + entrySize > segmentSize) {
95.             // Roll-over the segment (outside the read-lock)
96.         } else {
97.             // Copy entry into read cache segment
98.             cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99.                 entry.readableBytes());
100.             cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101.             return;
102.         }
103.     } finally {
104.         lock.readLock().unlock();
105.     }
106.
107.     // We could not insert in segment, we to get the write lock and roll-over to
108.     // next segment
109.     lock.writeLock().lock();
110.
111.     try {
112.         int offset = currentSegmentOffset.getAndAdd(entrySize);
113.         if (offset + entrySize > segmentSize) {
114.             // Rollover to next segment
115.             currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116.             currentSegmentOffset.set(alignedSize);
117.             cacheIndexes.get(currentSegmentIdx).clear();
118.             offset = 0;
119.         }
120.
121.         // Copy entry into read cache segment
122.         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(), entry.readableBytes());
123.         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
124.     } finally {
125.         lock.writeLock().unlock();
126.     }
127. }

```

Figura 6 – Mutation testing put

```

86 public void put(long ledgerId, long entryId, ByteBuf entry) {
87     int entrySize = entry.readableBytes();
88     int alignedSize = align64(entrySize);
89
90 1 lock.readLock().lock();
91
92     try {
93         int offset = currentSegmentOffset.getAndAdd(alignedSize);
94 3 if (offset + entrySize > segmentSize) {
95         // Roll-over the segment (outside the read-lock)
96     } else {
97         // Copy entry into read cache segment
98         cacheSegments.get(currentSegmentIdx).setBytes(offset, entry, entry.readerIndex(),
99             entry.readableBytes());
100         cacheIndexes.get(currentSegmentIdx).put(ledgerId, entryId, offset, entrySize);
101         return;
102     }
103 } finally {
104 1 lock.readLock().unlock();
105 }
106
107 // We could not insert in segment, we to get the write lock and roll-over to
108 // next segment
109 1 lock.writeLock().lock();
110
111     try {
112         int offset = currentSegmentOffset.getAndAdd(entrySize);
113 3 if (offset + entrySize > segmentSize) {
114         // Rollover to next segment
115 2 currentSegmentIdx = (currentSegmentIdx + 1) % cacheSegments.size();
116 1 currentSegmentOffset.set(alignedSize);
117 1 cacheIndexes.get(currentSegmentIdx).clear();
118         offset = 0;
119     }

```

Figura 7 - Coverage get

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
ReadCache(ByteBufAllocator, long, int)		100%		100%	0	2	0
get(long, long)		100%		100%	0	3	0
count()		100%		100%	0	2	0
ReadCache(ByteBufAllocator, long)		100%	n/a	n/a	0	1	0
close()		100%	n/a	n/a	0	1	0
size()		87%		83%	1	4	1

Figura 8 – Analisi coverage get

```

129. public ByteBuf get(long ledgerId, long entryId) {
130.     lock.readLock().lock();
131.
132.     try {
133.         // We need to check all the segments, starting from the current one and looking
134.         // backward to minimize the
135.         // checks for recently inserted entries
136.         int size = cacheSegments.size();
137.         for (int i = 0; i < size; i++) {
138.             int segmentIdx = (currentSegmentIdx + (size - i)) % size;
139.
140.             LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
141.             if (res != null) {
142.                 int entryOffset = (int) res.first;
143.                 int entryLen = (int) res.second;
144.
145.                 ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
146.                 entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
147.                 return entry;
148.             }
149.         }
150.     } finally {
151.         lock.readLock().unlock();
152.     }
153.
154.     // Entry not found in any segment
155.     return null;
156. }

```

Figura 9 - Mutation testing get

```

129. public ByteBuf get(long ledgerId, long entryId) {
130.     lock.readLock().lock();
131.
132.     try {
133.         // We need to check all the segments, starting from the current one and looking
134.         // backward to minimize the
135.         // checks for recently inserted entries
136.         int size = cacheSegments.size();
137.         for (int i = 0; i < size; i++) {
138.             int segmentIdx = (currentSegmentIdx + (size - i)) % size;
139.
140.             LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
141.             if (res != null) {
142.                 int entryOffset = (int) res.first;
143.                 int entryLen = (int) res.second;
144.
145.                 ByteBuf entry = allocator.directBuffer(entryLen, entryLen);
146.                 entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
147.                 return entry;
148.             }
149.         }
150.     } finally {
151.         lock.readLock().unlock();
152.     }

```

Figura 10 – Coverage write

write(ByteBuf)	<div><div></div></div>	74%	<div><div></div></div>	60%	3	6	5	21	0	1	
forceWrite(boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	7	7	1	1	
close()	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	6	6	1	1	
flushAndForceWrite(boolean)	<div><div></div></div>	0%		n/a	1	1	3	3	1	1	
flushAndForceWriteIfRegularFlush(boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	3	3	1	1	
clear()	<div><div></div></div>	0%		n/a	1	1	3	3	1	1	
getFileChannelPosition()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
getNumOfBytesInWriteBuffer()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
getUnpersistedBytes()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
Total		128 of 394	67%	19 of 38	50%	22	34	35	94	8	15

Figura 11 – Analisi coverage write

```

119.     public void write(ByteBuf src) throws IOException {
120.         int copied = 0;
121.         boolean shouldForceWrite = false;
122.         synchronized (this) {
123.             int len = src.readableBytes();
124.             // System.out.println("len: "+len);
125.             while (copied < len) {
126.                 int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
127.                 // System.out.println("bytesToCopy: "+bytesToCopy);
128.                 writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
129.                 // byte[] dest = new byte [bytesToCopy];
130.                 // writeBuffer.getBytes(0, dest);
131.                 // System.out.println(Arrays.toString(dest));
132.                 copied += bytesToCopy;
133.
134.                 // if we have run out of buffer space, we should flush to the
135.                 // file
136.                 if (!writeBuffer.isWritable()) {
137.                     flush();
138.                 }
139.             }
140.             position += copied;
141.             if (doRegularFlushes) {
142.                 unpersistedBytes.addAndGet(copied);
143.                 if (unpersistedBytes.get() >= unpersistedBytesBound) {
144.                     flush();
145.                     shouldForceWrite = true;
146.                 }
147.             }
148.             if (shouldForceWrite) {
149.                 forceWrite(false);
150.             }
151.         }
152.     }

```

Figura 12 – Coverage write (dopo miglioramento della test suite)













Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● write(ByteBuf)		100%		100%	0	6	0	21	0	1
● BufferedChannel(ByteBufAllocator, FileChannel, int, int, long)		100%		100%	0	2	0	11	0	1
● forceWrite(boolean)		100%		50%	1	2	0	7	0	1
● flush()		100%		50%	1	2	0	6	0	1
● BufferedChannel(ByteBufAllocator, FileChannel, int, long)		100%		n/a	0	1	0	2	0	1
● BufferedChannel(ByteBufAllocator, FileChannel, int)		100%		n/a	0	1	0	2	0	1
● getFileChannelPosition()		100%		n/a	0	1	0	1	0	1
● position()		100%		n/a	0	1	0	1	0	1

Figura 13 – Mutation testing write

```

119     public void write(ByteBuf src) throws IOException {
120         int copied = 0;
121         boolean shouldForceWrite = false;
122         synchronized (this) {
123             int len = src.readableBytes();
124             while (copied < len) {
125                 int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
126                 writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
127                 copied += bytesToCopy;
128
129                 // if we have run out of buffer space, we should flush to the
130                 // file
131                 if (!writeBuffer.isWritable()) {
132                     flush();
133                 }
134
135                 position += copied;
136                 if (doRegularFlushes) {
137                     unpersistedBytes.addAndGet(copied);
138                     if (unpersistedBytes.get() >= unpersistedBytesBound) {
139                         flush();
140                         shouldForceWrite = true;
141                     }
142                 }
143             }
144             if (shouldForceWrite) {
145                 forceWrite(false);
146             }
147         }
148     }

```

Figura 14 – Coverage read

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
flush()		100%		50%	1	2	0	6	0	1
BufferedChannel(ByteBufAllocator, FileChannel, int, long)		100%		n/a	0	1	0	2	0	1
BufferedChannel(ByteBufAllocator, FileChannel, int)		100%		n/a	0	1	0	2	0	1
position()		100%		n/a	0	1	0	1	0	1
BufferedChannel(ByteBufAllocator, FileChannel, int, int, long)		95%		50%	1	2	0	11	0	1
read(ByteBuf, long, int)		79%		61%	6	10	5	26	0	1

Figura 15 – Analisi coverage read

```

251.     @Override
252.     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
253.         long prevPos = pos;
254.         while (length > 0) {
255.             // check if it is in the write buffer
256.             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
257.                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
258.                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
259.
260.                 if (bytesToCopy == 0) {
261.                     throw new IOException("Read past EOF");
262.                 }
263.
264.                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
265.                 pos += bytesToCopy;
266.                 length -= bytesToCopy;
267.             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
268.                 // here we reach the end
269.                 break;
270.                 // first check if there is anything we can grab from the readBuffer
271.             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
272.                 int positionInBuffer = (int) (pos - readBufferStartPosition);
273.                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
274.                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
275.                 pos += bytesToCopy;
276.                 length -= bytesToCopy;
277.                 // let's read it
278.             } else {
279.                 readBufferStartPosition = pos;
280.
281.                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
282.                     readBufferStartPosition);
283.                 if (readBytes <= 0) {
284.                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
285.                 }
286.                 readBuffer.writerIndex(readBytes);
287.             }
288.         }
289.         return (int) (pos - prevPos);
290.     }

```

Figura 16 – Coverage read (dopo miglioramento della test suite)

● read(ByteBuf, long, int)	<div><div></div></div>	91%	<div><div></div></div>	66%	5	10	2	26	0	1	
● write(ByteBuf)	<div><div></div></div>	74%	<div><div></div></div>	60%	3	6	5	19	0	1	
● forceWrite(boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	6	6	1	1	
● flushAndForceWrite(boolean)	<div><div></div></div>	0%		n/a	1	1	3	3	1	1	
● flushAndForceWriteIfRegularFlush(boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	3	3	1	1	
● getFileChannelPosition()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
● getNumOfBytesInWriteBuffer()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
● getUnpersistedBytes()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
● position()	<div><div></div></div>	0%		n/a	1	1	1	1	1	1	
Total		94 of 394	76%	17 of 38	55%	20	34	24	91	7	15

Figura 17 – Analisi coverage read (dopo miglioramento della test suite)

```

246.     @Override
247.     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
248.         long prevPos = pos;
249.         while (length > 0) {
250.             // check if it is in the write buffer
251.             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
252.                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
253.                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
254.
255.                 if (bytesToCopy == 0) {
256.                     throw new IOException("Read past EOF");
257.                 }
258.
259.                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
260.                 pos += bytesToCopy;
261.                 length -= bytesToCopy;
262.             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
263.                 // here we reach the end
264.                 break;
265.                 // first check if there is anything we can grab from the readBuffer
266.             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
267.                 int positionInBuffer = (int) (pos - readBufferStartPosition);
268.                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
269.                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
270.                 pos += bytesToCopy;
271.                 length -= bytesToCopy;
272.                 // let's read it
273.             } else {
274.                 readBufferStartPosition = pos;
275.
276.                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
277.                     readBufferStartPosition);
278.                 if (readBytes <= 0) {
279.                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
280.                 }
281.                 readBuffer.writerIndex(readBytes);
282.             }
283.         }
284.         return (int) (pos - prevPos);
285.     }

```


Figura 18 – Mutation testing read

```

246 @Override
247 public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
248     long prevPos = pos;
249     while (length > 0) {
250         // check if it is in the write buffer
251         if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
252             int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
253             int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
254
255             if (bytesToCopy == 0) {
256                 throw new IOException("Read past EOF");
257             }
258
259             dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
260             pos += bytesToCopy;
261             length -= bytesToCopy;
262         } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
263             // here we reach the end
264             break;
265             // first check if there is anything we can grab from the readBuffer
266         } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
267             int positionInBuffer = (int) (pos - readBufferStartPosition);
268             int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
269             dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
270             pos += bytesToCopy;
271             length -= bytesToCopy;
272             // let's read it
273         } else {
274             readBufferStartPosition = pos;
275
276             int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
277                 readBufferStartPosition);
278             if (readBytes <= 0) {
279                 throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
280             }
281             readBuffer.writerIndex(readBytes);
282         }
283     }
284     return (int) (pos - prevPos);
285 }

```

Figura 19 - Report pit 2

```

246 @Override
247 public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
248     long prevPos = pos;
249     while (length > 0) {
250         // check if it is in the write buffer
251         if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
252             int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
253             int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
254
255             if (bytesToCopy == 0) {
256                 throw new IOException("Read past EOF");
257             }
258
259             dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
260             pos += bytesToCopy;
261             length -= bytesToCopy;
262         } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
263             // here we reach the end
264             break;
265             // first check if there is anything we can grab from the readBuffer
266         } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
267             int positionInBuffer = (int) (pos - readBufferStartPosition);
268             int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
269             dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
270             pos += bytesToCopy;
271             length -= bytesToCopy;
272             // let's read it
273         } else {
274             readBufferStartPosition = pos;
275
276             int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
277                 readBufferStartPosition);
278             if (readBytes <= 0) {
279                 throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
280             }
281             readBuffer.writerIndex(readBytes);
282         }
283     }
284     return (int) (pos - prevPos);
285 }

```

Figura 20 – Coverage read (dopo miglioramento mutation testing)

● read(ByteBuf, long, int)	<div><div></div></div>	95%	<div><div></div></div>	72%	4	10	1	26	0	1	
● close()	<div><div></div></div>	92%	<div><div></div></div>	50%	1	2	1	6	0	1	
● flushAndForceWrite(boolean)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	3	3	1	1	
● flushAndForceWriteIfRegularFlush(boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	3	3	1	1	
● getFileChannelPosition()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1	
● getNumOfBytesInWriteBuffer()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1	
● getUnpersistedBytes()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1	
Total		34 of 394	91%	10 of 38	73%	13	34	11	94	5	15

Figura 21 – Analisi coverage read (dopo mutation testing)

```

246.     @Override
247.     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
248.         long prevPos = pos;
249.         while (length > 0) {
250.             // check if it is in the write buffer
251.             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
252.                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
253.                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
254.
255.                 if (bytesToCopy == 0) {
256.                     throw new IOException("Read past EOF");
257.                 }
258.
259.                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
260.                 pos += bytesToCopy;
261.                 length -= bytesToCopy;
262.             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
263.                 // here we reach the end
264.                 break;
265.                 // first check if there is anything we can grab from the readBuffer
266.             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
267.                 int positionInBuffer = (int) (pos - readBufferStartPosition);
268.                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
269.                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
270.                 pos += bytesToCopy;
271.                 length -= bytesToCopy;
272.                 // let's read it
273.             } else {
274.                 readBufferStartPosition = pos;
275.
276.                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
277.                     readBufferStartPosition);
278.                 if (readBytes <= 0) {
279.                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
280.                 }
281.                 readBuffer.writerIndex(readBytes);
282.             }
283.         }
284.         return (int) (pos - prevPos);
285.     }

```

Figura 22 – Coverage del metodo isValid

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● isValid(Group_ConstraintValidatorContext)	<div><div></div></div>	50%	<div><div></div></div>	50%	8	11	16	29	0	1
● GroupValidator()	<div><div></div></div>	100%		n/a	0	1	0	1	0	1
Total	53 of 109	51%	10 of 20	50%	8	12	16	30	0	2

Figura 23 – Analisi coverage del metodo isValid

```

30. public class GroupValidator extends AbstractValidator<GroupCheck, Group> {
31.
32.     @Override
33.     public boolean isValid(final Group group, final ConstraintValidatorContext context) {
34.         context.disableDefaultConstraintViolation();
35.
36.         boolean isValid = true;
37.
38.         if (group.getUserOwner() != null && group.getGroupOwner() != null) {
39.             isValid = false;
40.
41.             context.buildConstraintViolationWithTemplate(
42.                 getTemplate(EntityViolationType.InvalidGroupOwner,
43.                     "A group must either be owned by an user or a group, not both"));
44.             addPropertyNode("owner").addConstraintViolation();
45.         }
46.
47.         if (isValid && (group.getName() == null || !KEY_PATTERN.matcher(group.getName()).matches())) {
48.             isValid = false;
49.
50.             context.buildConstraintViolationWithTemplate(
51.                 getTemplate(EntityViolationType.InvalidName, group.getName()));
52.             addPropertyNode("name").addConstraintViolation();
53.         }
54.
55.         if (isValid) {
56.             Set<AnyType> anyTypes = new HashSet<>();
57.             for (ADynGroupMembership memb : group.getADynMemberships()) {
58.                 anyTypes.add(memb.getAnyType());
59.
60.                 if (memb.getAnyType().getKind() != AnyTypeKind.ANY_OBJECT) {
61.                     isValid = false;
62.
63.                     context.buildConstraintViolationWithTemplate(
64.                         getTemplate(EntityViolationType.InvalidADynMemberships,
65.                             "No user or group dynamic membership condition are allowed here"));
66.                     addPropertyNode("aDynMemberships").addConstraintViolation();
67.                 }
68.             }
69.
70.             if (isValid && anyTypes.size() < group.getADynMemberships().size()) {
71.                 context.buildConstraintViolationWithTemplate(
72.                     getTemplate(EntityViolationType.InvalidADynMemberships,
73.                         "Each dynamic membership condition requires a different "
74.                         + AnyType.class.getSimpleName()));
75.                 addPropertyNode("aDynMemberships").addConstraintViolation();
76.                 return false;
77.             }
78.
79.         }
80.
81.         return isValid;
82.     }

```

Figura 24 – Coverage del metodo isValid (dopo il miglioramento della testSuite)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● isValid(Group, ConstraintValidatorContext)	<div></div>	100%	<div></div>	100%	0	11	0	29	0	1
● GroupValidator()	<div></div>	100%		n/a	0	1	0	1	0	1
Total	0 of 109	100%	0 of 20	100%	0	12	0	30	0	2

Figura 25 – Mutation testing del metodo isValid (dopo miglioramento della test suite)

```

32  @Override
33  public boolean isValid(final Group group, final ConstraintValidatorContext context) {
34  1  context.disableDefaultConstraintViolation();
35
36  boolean isValid = true;
37
38  2  if (group.getUserOwner() != null && group.getGroupOwner() != null) {
39  isValid = false;
40
41  context.buildConstraintViolationWithTemplate(
42  getTemplate(EntityViolationType.InvalidGroupOwner,
43  "A group must either be owned by an user or a group, not both"));
44  addPropertyNode("owner").addConstraintViolation();
45  }
46
47  3  if (isValid && (group.getName() == null || !KEY_PATTERN.matcher(group.getName()).matches())) {
48  isValid = false;
49
50
51  context.buildConstraintViolationWithTemplate(
52  getTemplate(EntityViolationType.InvalidName, group.getName()));
53  addPropertyNode("name").addConstraintViolation();
54  }
55
56  1  if (isValid) {
57  Set<AnyType> anyTypes = new HashSet<>();
58  for (ADynGroupMembership memb : group.getADynMemberships()) {
59  anyTypes.add(memb.getAnyType());
60
61  1  if (memb.getAnyType().getKind() != AnyTypeKind.ANY_OBJECT) {
62  isValid = false;
63
64  context.buildConstraintViolationWithTemplate(
65  getTemplate(EntityViolationType.InvalidADynMemberships,
66  "No user or group dynamic membership condition are allowed here"));
67  addPropertyNode("aDynMemberships").addConstraintViolation();
68  }
69  }
70
71  3  if (isValid && anyTypes.size() < group.getADynMemberships().size()) {
72  context.buildConstraintViolationWithTemplate(
73  getTemplate(EntityViolationType.InvalidADynMemberships,
74  "Each dynamic membership condition requires a different "
75  + AnyType.class.getSimpleName()));
76  addPropertyNode("aDynMemberships").addConstraintViolation();
77  1  return false;
78  }

```

Figura 26 – Coverage del metodo getGroupOwnerRealm

● getGroupOwnerRealm(String, String)	<div><div></div></div>	100%		n/a	0	1	0	1	0	1
Total	0 of 181	100%	1 of 20	95%	1	16	0	37	0	6

Figura 27 – Analisi coverage del metodo getGroupOwnerRealm

```

32.  public static String getGroupOwnerRealm(final String realmPath, final String groupKey) {
33.  1  return realmPath + '@' + groupKey;
34.  }

```

Figura 28 – Mutation testing sul metodo getGroupOwnerRealm

```

32  public static String getGroupOwnerRealm(final String realmPath, final String groupKey) {
33  1  return realmPath + '@' + groupKey;
34  }

```

Figura 29 - Coverage del metodo parseGroupOwnerRealm

● parseGroupOwnerRealm(String)	<div><div></div></div>	100%	<div><div></div></div>	75%	1	3	0	4	0	1
--	------------------------	------	------------------------	-----	---	---	---	---	---	---

Figura 30 – Analisi coverage del metodo parseGroupOwnerRealm

```

36.     public static Optional<Pair<String, String>> parseGroupOwnerRealm(final String input) {
37.         String[] split = input.split("@");
38.         return split == null || split.length < 2
39.             ? Optional.empty()
40.             : Optional.of(Pair.of(split[0], split[1]));
41.     }

```

Figura 31 – Mutation testing del metodo parseGroupOwnerRealm

```

36     public static Optional<Pair<String, String>> parseGroupOwnerRealm(final String input) {
37         String[] split = input.split("@");
38 4  return split == null || split.length < 2
39         ? Optional.empty()
40         : Optional.of(Pair.of(split[0], split[1]));
41     }

```

Figura 32 – Coverage del metodo normalizingAddTo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● normalizingAddTo(Set, String)	<div></div>	100%	<div></div>	100%	0	6	0	11	0	1

Figura 33 – Analisi coverage del metodo normalizingAddTo

```

43.     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm) {
44.         boolean dontAdd = false;
45.         Set<String> toRemove = new HashSet<>();
46.         for (String realm : realms) {
47.             if (newRealm.startsWith(realm)) {
48.                 dontAdd = true;
49.             } else if (realm.startsWith(newRealm)) {
50.                 toRemove.add(realm);
51.             }
52.         }
53.
54.         realms.removeAll(toRemove);
55.         if (!dontAdd) {
56.             realms.add(newRealm);
57.         }
58.         return !dontAdd;
59.     }

```

Figura 34 – Mutation testing sul metodo normalizingAddTo

```

43     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm) {
44         boolean dontAdd = false;
45         Set<String> toRemove = new HashSet<>();
46         for (String realm : realms) {
47 1  if (newRealm.startsWith(realm)) {
48             dontAdd = true;
49 1  } else if (realm.startsWith(newRealm)) {
50             toRemove.add(realm);
51         }
52     }
53
54     realms.removeAll(toRemove);
55 1  if (!dontAdd) {
56         realms.add(newRealm);
57     }
58 2  return !dontAdd;
59 }

```

Figura 35 – Coverage del metodo normalize

● normalize(Collection)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	5	0	1
---	------------------------	------	------------------------	------	---	---	---	---	---	---

Figura 36 – Analisi coverage del metodo normalize

```

61.     public static Pair<Set<String>, Set<String>> normalize(final Collection<String> realms) {
62.         Set<String> normalized = new HashSet<>();
63.         Set<String> groupOwnership = new HashSet<>();
64.         if (realms != null) {
65.             realms.forEach(realm -> {
66.                 if (realm.indexOf('@') == -1) {
67.                     normalizingAddTo(normalized, realm);
68.                 } else {
69.                     groupOwnership.add(realm);
70.                 }
71.             });
72.         }
73.
74.         return Pair.of(normalized, groupOwnership);
75.     }

```

Figura 37 – Mutation testing sul metodo normalize

```

61     public static Pair<Set<String>, Set<String>> normalize(final Collection<String> realms) {
62         Set<String> normalized = new HashSet<>();
63         Set<String> groupOwnership = new HashSet<>();
64     1     if (realms != null) {
65     1         realms.forEach(realm -> {
66     1             if (realm.indexOf('@') == -1) {
67                 normalizingAddTo(normalized, realm);
68             } else {
69                 groupOwnership.add(realm);
70             }
71         });
72     }
73
74     1     return Pair.of(normalized, groupOwnership);
75 }

```

Figura 38 – Coverage del metodo getEffective

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● getEffective(Set, String)	<div><div></div></div>	100%	<div><div></div></div>	100%	0	2	0	11	0	1

Figura 39 – Analisi coverage del metodo `getEffective`

```
99.     public static Set<String> getEffective(final Set<String> allowedRealms, final String requestedRealm) {
100.         Pair<Set<String>, Set<String>> normalized = normalize(allowedRealms);
101.
102.         Collection<String> requested = Arrays.asList(requestedRealm);
103.
104.         Set<String> effective = new HashSet<>();
105.         effective.addAll(requested.stream().
106.             filter(new StartsWithPredicate(normalized.getLeft())).collect(Collectors.toSet()));
107.         effective.addAll(normalized.getLeft().stream().
108.             filter(new StartsWithPredicate(requested)).collect(Collectors.toSet()));
109.
110.         // includes group ownership
111.         effective.addAll(normalized.getRight());
112.
113.         // includes dynamic realms
114.         if (allowedRealms != null) {
115.             effective.addAll(allowedRealms.stream().filter(new DynRealmsPredicate()).collect(Collectors.toSet()));
116.         }
117.
118.         return effective;
119.     }
```

Figura 40 – Mutation testing sul metodo `getEffective`

```
99     public static Set<String> getEffective(final Set<String> allowedRealms, final String requestedRealm) {
100         Pair<Set<String>, Set<String>> normalized = normalize(allowedRealms);
101
102         Collection<String> requested = Arrays.asList(requestedRealm);
103
104         Set<String> effective = new HashSet<>();
105         effective.addAll(requested.stream().
106             filter(new StartsWithPredicate(normalized.getLeft())).collect(Collectors.toSet()));
107         effective.addAll(normalized.getLeft().stream().
108             filter(new StartsWithPredicate(requested)).collect(Collectors.toSet()));
109
110         // includes group ownership
111         effective.addAll(normalized.getRight());
112
113         // includes dynamic realms
114 1   if (allowedRealms != null) {
115             effective.addAll(allowedRealms.stream().filter(new DynRealmsPredicate()).collect(Collectors.toSet()));
116         }
117
118 1   return effective;
119     }
```