ISW2 Relazione Software Testing

Pier Francesco Contino

Giugno 2020

Bookkeeper

Cookie.java

Per la prima classe da sottoporre al testing è stata scelta la classe **Cookie** in quanto in base ad un'analisi effettuata essa presenta un'età tra le più alte in congiunzione con avere la maggior parte delle modifiche effettuate nelle versioni più remote, questo lascerebbe pensare che la maggior parte del codice di questa classe sia antico. Lo scopo della classe Cookie è quella di salvare le informazioni relative ad un Bookie (il server che salva i Ledger) così che la sua configurazione rimanga la stessa durante la sua vita.
Prima di iniziare con i test osserviamo gli attributi che compongono lo stato dell'oggetto Cookie:

```
private final int layoutVersion;
private final String bookieHost;
private final String journalDirs;
private final String ledgerDirs;
private final String instanceId;
```

Per il campo layoutVersion dato che è una versione ci si aspetta che sia non negativo, vedendo il codice scopriamo che l'ultima versione è la 4.
I campi ledgerDirs e journalDirs sono delle stringhe in cui vengono concatenate delle directory utilizzando un carattere separatore definito come `\t`.
I campi instanceId e bookieHost sono delle stringhe contenenti degli identificativi.

`public static String encodeDirPaths(String[] dirs)`

Dato che questo metodo ha come unico argomento un vettore di stringhe sono state scelte le seguenti classi di equivalenza.

Parametro	Classi equivalenza		
dirs	null	vettore con dimensione 0	vettore con valori

Effettuando un test su ogni classe di equivalenza abbiamo ottenuto come risultati rispettivamente un'eccezione per la prima classe di equivalenza, una stringa che rappresenta secondo la convezione del metodo l'assenza di directory per la seconda classe e un risultato legittimo per l'ultima classe di equivalenza.

Parametro	Valori Test		
dirs	null	String[0]	`{"dir1", "dir2"}`
Risultati	eccezione	`"0"`	`"2\tdir 1\tdir 2"`

`String[] getLedgerDirPathsFromCookie()`

Anche se dalla firma questo metodo potrebbe utilizzare l'intero stato dell'oggetto Cookie vedendo il nome e andando a vedere il codice possiamo vedere che utilizza solo l'attributo *ledgerDirs*, quindi si è scelto di utilizzare solo quello per la generazione delle classi di equivalenza.

Parametro	Classi equivalenza		
ledgerDirs	null	stringa vuota	stringa conforme

Osservando i risultati dei test abbiamo un'eccezione causato dal parametro *ledgerDirs* = *null*, un vettore con dimensione = 0 causato dal parametro *ledgerDirs* = *""* e un risultato valido per la stringa con il formato conforme al metodo.

Parametro	Valori Test		
ledgerDirs	null	""	"2\tdir 1\tdir 2"
Risultati	eccezione	String[0]	{"dir1", "dir2"}

`public void verify(Cookie c)throws BookieException.InvalidCookieException`

Questo metodo utilizza due oggetti complessi Cookie l'argomento del metodo e l'oggetto da cui viene chiamato il metodo, esso verifica se essi siano congrui. Sempre estrapondo informazioni dal codice vediamo che congrui si intende che il Cookie c abbia la versione maggiore di 3 e che abbiano journalDirs, bookkieHost, ledgerDirs. InstanceId è l'unico parametro dove viene fatto un controllo per verificare che l'oggetto sia diverso da null prima di usare il metodo equals (il quale risulterebbe in un'eccezione).

Parametro	Classi equivalenza			
layoutVersion				
bookieHost	null	stringa vuota	stringa conforme	
journalDirs	null	stringa vuota	stringa conforme	
ledgerDirs	null	stringa vuota	stringa conforme	
instanceID	null	stringa vuota	stringa conforme	
c.layoutVersion	< 3	> 3		
c.bookieHost	= bookieHost	!= bookieHost		
c.journalDirs	= journalDirs	!= journalDirs		
c.ledgerDirs	= ledgerDirs	!= ledgerDirs		
c.instanceID	= instanceID	!= instanceID		

Si è scelto di utilizzare un criterio di testing unidimensionale andando così a coprire tutte le classi di equivalenza ma non la combinazione di esse. La condizione di testing utilizzata è stata la presenza di un'eccezione dato che il metodo è void.

oggetto	layoutVersion	bookieHost	journalDirs	ledgerDirs	instanceID
0	2	null	null	null	null
1	3	""	""	""	""
2	3	"192.168.1.1:80"	"2\tdir 1\tdir 2"	"2\tdir 1\tdir 2"	"id"

c	this	Eccezione
oggetto 0	oggetto 0	true
oggetto 0	oggetto 2	true
oggetto 1	oggetto 1	false
oggetto 1	oggetto 2	true
oggetto 2	oggetto 2	false
oggetto 2	oggetto 1	true

I test sono andati a buon fine anche se vale la pena sottolineare che il metodo considera valido il match di stringhe vuote per ogni parametro.

`public void verifyIsSuperSet(Cookie c)throws BookieException.InvalidCookieException`

Questo metodo è molto simile al precedente solo che considera validi i Cookie anche quando il parametro ledgerDirs non è uguale ma si ha che il primo è un sovrainsieme del secondo.

Parametro	Classi equivalenza			
layoutVersion				
bookieHost	null	stringa vuota	stringa conforme	
journalDirs	null	stringa vuota	stringa conforme	
ledgerDirs	null	stringa vuota	stringa conforme	
instanceID	null	stringa vuota	stringa conforme	
c.layoutVersion	<3	>=3		
c.bookieHost	= bookieHost	!= bookieHost		
c.journalDirs	= journalDirs	!= journalDirs		
c.ledgerDirs	= ledgerDirs	!= ledgerDirs	superset	subset
c.instanceID	= instanceID	!= instanceID		

Anche qui tutti i test posti hanno risultati congrui con quanto ci si aspettava.

Metriche di accuratezza dei test

Scegliamo come metriche di accuratezza dei test la condition coverage e la statement coverage (relative alla classe testata), per misurarle utilizziamo Sonarcloud. Abbiamo quindi una condition coverage del 38,2 % e una statement coverage del 37 % (indicata come line coverage su Sonarcloud). Abbiamo come obbiettivo il miglioramento delle metriche per arrivare ad un 50 % di coverage per entrambe.

Per farlo andremo a modificare il metodo CookieVeryfyTest per coprire tutto gli statement e le condition chiamate dal metodo CookieVerify aggiungendo ulteriori parametri, questi parametri non sarebbero stati necessari se fosse stato usato un criterio di testing multidimensionale

```
        };

        return Arrays.asList(new Object[][] {
                {cookies[0], cookies[0], true},
                {cookies[0], cookies[2], true},
                {cookies[1], cookies[1], false},
                {cookies[1], cookies[2], true},
                {cookies[2], cookies[2], false},
                {cookies[2], cookies[1], true},

                //added after coverage analysis
                {cookies[1], cookies[0], true},
                {getCookie(3, valid_host, "2\tdir1\tdir2",
                    "2\tdir1\tdir2", id1),
                getCookie(3, valid_host, "2\tdir1\tdir2", "2\tdir1\tdir2",
                    id2), true},
```

Poi andremo a testare altri 2 metodi:
`public boolean isBookieHostCreatedFromIp()throws IOException` che ritorna se il bookie host è stato creato da un indirizzo ip o se viceversa è stato creato da un URL

```
    public void testIsBookieHostCreatedFromIp() throws IOException {
        builder.setBookieHost("192.168.1.1:80");
        Cookie fromIp = builder.build();
        assertTrue(fromIp.isBookieHostCreatedFromIp());
        builder.setBookieHost("www.google.com:80");
        Cookie fromURL = builder.build();
        assertFalse(fromURL.isBookieHostCreatedFromIp());

        try {
            builder.setBookieHost("msg");
            builder.build().isBookieHostCreatedFromIp();
            fail();
        } catch (IOException e) {
            System.out.println("Expected exception: " + e.getMessage());
        }
```

```
        try {
            builder.setBookieHost("192.168.1.1:nAn");
            builder.build().isBookieHostCreatedFromIp();
            fail();
        } catch (IOException e) {
            System.out.println("Expected exception: " + e.getMessage());
        }

    }
```

`public String toString()` che ritorna la rappresentazione stringa dell'oggetto Cookie.

```
    @Test public void testToString() {
        Cookie cookie3 = getCookie(3, "192.168.1.1:80", "2\tdir1\tdir2",
                                   "2\tdir1\tdir2", "id");
        assertEquals("4\n192.168.1.1:80\n2\tdir1\tdir2\n2\tdir1\tdir2\n",
            cookie3.toString());
        Cookie cookie4 = getCookie(4, "192.168.1.1:80", "2\tdir1\tdir2",
                "2\tdir1\tdir2", "id");
        assertEquals("4\n" +
                "bookieHost: \"192.168.1.1:80\"\n" +
                "journalDir: \"2\\tdir1\\tdir2\"\n" +
                "ledgerDirs: \"2\\tdir1\\tdir2\"\n" +
                "instanceId: \"id\"\n", cookie4.toString());

    }
```

Il metodo toString ha la particolarità di stampare l'ultima versione del layout definita (la 4) anche se poi utilizza per la stampa il layout della versione effettiva.

Mutation

Utilizzando pit abbiamo ottenuto una mutation coverage del 45% Andando poi a vedere i metodi non coperti abbiamo aggiunto dei casi di test sempre per il metodo verify e riprovato l'analisi portandola al 47 %.

```
                //added after mutation analysis
                {getCookie(3, valid_host, "2\tdir1\tdir2",
                    "2\tdir1\tdir2", id1),
                getCookie(3, valid_host, "2\tdir1\tdir2", "2\tdir1\tdir3",
                    id1), true},

                {getCookie(3, valid_host, "2\tdir1\tdir2",
                    "2\tdir1\tdir2", id1),
                getCookie(2, valid_host,"2\tdir1\tdir2","2\tdir1\tdir3",
                    id2), true},

                {getCookie(2, valid_host, "2\tdir1\tdir2",
                    "2\tdir1\tdir3", id2),
                getCookie(2, valid_host,"2\tdir1\tdir2","2\tdir1\tdir3",
                    id2), true}
        });
    }
```

WeightedRandomSelectionImpl

Questa classe ha lo scopo di selezionare un numero casuale tra quelli inseriti, i numeri casuali hanno un peso il quale favorisce la selezione, questa classe è l'implementazione dell'interfaccia WeightedRandomSelection.
La classe ha un costruttore semplice che setta solo il parametro maxProbabilityMultiplier che poi verrà

utilizzato nelle altre operazioni, il metodo principale è updateMap che carica i dati all'interno dell'oggetto e effettua delle computazioni per settare gli attributi necessari alla selezione del numero random, questi attributi hanno visibilità all'interno del package. Si è deciso quindi di sfruttare l'accessibilità dello stato per testare i metodi separatamente.

```
public void updateMap(Map<T, WeightedObject> map)
```

Parametro	Classi equivalenza			
Map	vuota	oggetti con peso 0	oggetti con peso omogeneo	oggetti con peso disomogeneo
maxProbabilityMultiplier	-1	0	!= 0	

```
public T getNextRandom()
```

Parametro	Classi equivalenza			
cummulativeMap	vuota	un elemento	elementi egual distanziati	elementi non egual distanziati
randomMax	congruo			

Qui i parametri sono parte dello stato che verrebbe inizializzato da updateMap, nel test si è scelto di settare direttamento lo stato dato.

Metriche di accuratezza dei test

Scegliamo come metriche di accuratezza dei test sempre la condition coverage e la statement coverage (relative alla classe testata), per misurarle utilizziamo Sonarcloud. Abbiamo quindi una condition coverage del 70 % e una statement coverage del 78,9 % (indicata come line coverage su Sonarcloud). Abbiamo come obbiettivo il miglioramento delle metriche per arrivare ad un 90 % di coverage per entrambe, dato che alcune parti del codice non possono essere coperte in quanto dipendono dallo stato del logger il quale è costante.

Per farlo andremo a testare gli altri metodi che sono dei setter e in più aggiungeremo un parametro a UpdateMapTest.

Codice UpdateMapTest

```
res1.put(null, 0.0);

map2.put("1", getWeightedObject(0));
map2.put("2", getWeightedObject(0));
res2.put("1", 0.0);
res2.put("2", 0.5);

map3.put("1", getWeightedObject(2));
map3.put("2", getWeightedObject(2));
map3.put("3", getWeightedObject(2));
res3.put("1", 0.0);
res3.put("2", 0.3333333333333333);
res3.put("3", 0.6666666666666666);

map4.put("1", getWeightedObject(1));
map4.put("2", getWeightedObject(2));
map4.put("3", getWeightedObject(3));
res4.put("1", 0.0 );
res4.put("2", 1.0 / 6.0);
res4.put("3", 1.0 / 6.0 + 2.0 / 6.0);

//added after coverage goal

map5.put("1", getWeightedObject(4));
map5.put("2", getWeightedObject(3));
```

```
map5.put("3", getWeightedObject(2));
map5.put("4", getWeightedObject(1));
res5.put("1", 0.0 );
res5.put("2", 0.25);
res5.put("3", 0.5);
res5.put("4", 0.7);



return Arrays.asList(new Object[][] {
        {-1, map1, res1},
        {-1, map2, res2},
        {-1, map3, res3},
        {-1, map4, res4},
        {0, map1, res1},
        {0, map2, res2},
        {0, map3, res3},
        {0, map4, res4},
        {1, map1, res1},
        {1, map2, res2},
        {1, map3, res3},
        {1, map4, res4},
```

Codice GetNextRandomTest

```
map2.put(0.0, "1");

map3.put(0.0, "1");
map3.put(1.0, "2");

map4.put(0.0, "1");
map4.put(1.0 / 6.0, "2");
map4.put(1.0 / 6.0 + 2.0 / 6.0, "3");

return Arrays.asList(new Object[][] {
        {0.0, map1},
        {0.0, map2},
        {1.0, map3},
        {1.0 / 6.0 + 2.0 / 6.0, map4}
});
```

Mutation

Per quanto riguarda la mutation coverage abbiamo un valore del 74%, proviamo ora a migliorare questo valore andando a vedere in quali casi i test non sono riusciti a "uccidere" le mutazioni.
Siamo riusciti ad uccidere una mutazione aggiungendo una verifica del rilascio del lock in lettura nel test di getNextRandom così arrivando a una coverage del 77%.

Zookeeper

AvgMinMaxCounter.java

Questa classe consiste in un contatore di numeri long che fornisce metodi per reperire la media, il totale, in numero di valori, il minimo e il massimo; l'aggiornamento di tali valori avviene in modo atomico, il costruttore permette di dare un nome alla metrica misurata.

`public void addDataPoint(long value)`

Questo metodo aggiunge un valore al contatore e chiama le funzioni interne per aggiornare il minimo e il massimo, quindi viene utilizzato nel setup dello stato dell'ogetto per il testing.

`public double getAvg()`

Questo metodo calcola e ritorna la media dei valori dati al contatore.

Parametro	Classi equivalenza		
count	$= 0$	$!= 0$	
total	< 0	$= 0$	> 0

I parametri sopra indicati sono gli attributi utilizzati dal metodo, per settare tali valori abbiamo generato l'oggetto e poi abbiamo inseriti dei valori congrui.

`public void testGetMin()`

Questo metodo aggiorna il minimo e poi lo restituisce, il valore restituito quando il contatore è vuoto è 0, ma lo stato interno è settato con il valore maggiore assumibile da un long.

Parametro	Classi equivalenza			
values	vuoto	valori $= 0$	valori > 0	valori < 0

Per il testing abbiamo scelto dei valori estremi di questa classe di equivalenza, abbiamo però deciso di non esplorare il caso di overflow.

`public void testGetMax()`

Per questo metodo abbiamo utilizzato gli stessi valori del precedente test, in quanto è strutturato nella stessa maniera

`public void reset()`

Questo metodo resetta il contatore allo stato vuoto.

`public void resetMax()`

Questo metodo resetta il valore massimo ma lascia gli altri valori integri, nell'implementazione il valore massimo viene settato al valore del minimo. Quest'implementazione può risultare in un malfunzionamento quando il valore minimo è Long.MIN, poichè in questo caso il max si comporta come quando il contatore è vuoto; quest'eventualità è stata testata e si è verificato un fallimento del test.
Anche se una simile eventualità è molto rara riteniamo sia un comportamento scorretto del metodo.
Questo caso di test è stato commentato in quanto il fallimento del test impedisce l'uso di pit per il mutation testing.
I restanti metodi della classe sono stati coperti ma non vengono riportati in quanto semplici setter e getter.

Metriche di accuratezza dei test

Scegliamo come metriche di accuratezza dei test sempre la condition coverage e la statement coverage (relative alla classe testata), per misurarle utilizziamo Sonarcloud. Abbiamo quindi una condition coverage del 85,7 % e una statement coverage del 100 % (indicata come line coverage su Sonarcloud). Vediamo che le uniche condizioni non coperte da test sono quelle sul fallimento dell'operazione atomica compareAndSet, per verificarle sarebbe necessario utilizzare dei test junit concorrenti, per questo motivo si è deciso di evitare il test di tali condizioni.

Mutation

Utilizzando pit vediamo che abbiamo una mutation coverage dell'84%, osservando il report vediamo che le uniche mutazioni che non vengono scoperte sono quelle relative alla condizione sul metodo compareAndSet ma come precedente illustrato aggiungere un test per quella condizione richiederebbe l'utilizzo della concorrenza.

Codice Test

```
public AvgMinMaxCounterTest(long[] values, double avg, long min, long
    max, long total, long count) {
     this.values = values;
     this.avg = avg;
     this.min = min;
     this.max = max;
     this.total = total;
     this.count = count;
}

@Parameterized.Parameters
public static Collection param() {
     return Arrays.asList(new Object[][] {
             {new long[] {}, 0.0, 0, 0, 0, 0},
             {new long[] {0}, 0.0, 0, 0, 0, 1},
             {new long[] {0, Long.MAX_VALUE}, Long.MAX_VALUE / 2 , 0,
                 Long.MAX_VALUE, Long.MAX_VALUE, 2},
             //{new long[] {0, Long.MIN_VALUE}, Long.MIN_VALUE / 2.0 ,
                 Long.MIN_VALUE, 0, Long.MIN_VALUE, 2}
             {new long[] {-1, -2}, -1.5 , -2, -1, -3, 2}
     });
}
```

Listing 1: Parametri utilizzati per tutti i test

CreateCommand.java

Questa classe rappresenta il comando cli di zookeeper per creare un nodo, estende la classe astratta CliCommand, ha un costruttore senza parametri, un metodo per il parsing degli argomenti del comando e un'altro che lancia il comando. Quest'ultimo metodo richiede di aver settato l'oggetto Zookeeper, il quale svolge l'attività di eseguire il comando, si è scelto quindi di utilizzare un mock per Zookeeper in quanto stiamo svolgendo un unit testing e ci interessa il corretto funzionamento della sola classe CreateCommand. I parametri di questo test vengono settati tramite il metodo di parsing che inizializza lo stato dell'oggetto settando il comando.

`public CliCommand parse(String[] cmdArgs)throws CliParseException`

Il metodo prende come input degli argomenti stringa che verranno parsati secondo la sintassi "create [-s] [-e] [-c] [-t ttl] path [data] [acl]" possiamo quindi dedurre che il numero minimo di argomenti è 2, inoltre possiamo testare come il metodo si comporterebbe immettendo un argomento non specificato dalla guida. Il valore di ritorno del metodo è lo stesso oggetto su cui viene chiamato il metodo, il metodo setta dei campi specifici dell'oggetto che però sono private quindi non possono essere osservati direttamente, quindi andremo ad osservare la presenza o no dell'eccezione.

Parametro	Classi equivalenza			
args	dimensione < 2	argomenti specificati	argomenti non specificati	stringhe vuote
arg [-t ttl]	ttl presente	ttl assente		

`public boolean exec()throws CliException`

Questo metodo non ha parametri ma si basa sullo stato dell'oggetto su cui viene chiamato, per settare lo stato abbiamo utilizzato il metodo precedente. Il valore di ritorno è sempre true quindi viene ignorato

nei test, ci baseremo anche qui sulla presenza o no dell'eccezione per verificare il funzionamento di questo metodo.
Per estrarre le classi di equivalenza ci baseremo sull'osservazione del parametro ttl (time to live) e su uno sguardo alla funzione che ci fornisce i parametri che non possono essere utilizzati congiuntamente.

Parametro	Classi equivalenza			
argomento [-t ttl]	ttl < 0	ttl $= 0$	ttl > 0	
argomento [-t ttl]	[-e]	[-c]		
argomento [-c]	[-e]	[-s]		

Metriche di accuratezza dei test

Scegliamo come metriche di accuratezza dei test sempre la condition coverage e la statement coverage (relative alla classe testata), per misurarle utilizziamo Sonarcloud. Abbiamo quindi una condition coverage del 90 % e una statement coverage del 79,4 % (indicata come line coverage su Sonarcloud.
Abbiamo come obbiettivo il raggiungimento del 100 % del coverage sia per gli statement che per le condizioni, per ottenerlo dobbiamo utilizzare una versione diversa del mock di Zookeeper così da poter lanciare eccezioni e in più dobbiamo aggiungere nuove combinazioni di argomenti.

Mutation

Utilizzando pit vediamo che abbiamo una mutation coverage dell'57% abbastanza bassa rispetto alla line coverage. Andando a vedere le mutazioni non coperte vediamo che molte si riferiscono ad un parametro passato al mock di Zookeeper, altre si rifeniscono alla modifica di messaggi mandati sullo stream standard error; quindi andiamo ad utilizzare una mock per l'error stream e nel test usiamo il metodo verify di mockito. Un'altra modifica fatta è la granularità della verifica dell'eccezione dove passiamo da un semplice stato booleano per la presenza dell'eccezione a uno dove testiamo la classe a cui appartiene l'eccezione. Infine inseriamo il controllo sui valori di ritorno precedentemente ignorati. Dopo queste modifiche abbiamo raggiunto il 100% di mutation coverage.

Codice Test

```
{new String[] {},  true},
{new String[] {"", ""}, false},
{new String[] {"create", "/zk_test"}, false},
{new String[] {"create", "/zk_test", "my_data"}, false},
{new String[] {"create", "/zk_test", "my_data", "-s"},
    false},
{new String[] {"create", "/zk_test", "my_data", "-e"},
    false},
{new String[] {"create", "/zk_test", "my_data", "-c"},
    false},
{new String[] {"create", "/zk_test", "my_data", "-t",
    "100"}, false},
{new String[] {"create", "/zk_test", "my_data", "-t"},
    true},
{new String[] {"create", "/zk_test", "my_data", "-l"},
    true}
```

Listing 2: Parametri utilizzati CreateCommandTest

```
{new String[] {"create", "/zk_test", "my_data"}, null, null},
{new String[] {"create", "/zk_test", "my_data", "-s"}, null,
    null},
{new String[] {"create", "/zk_test", "my_data", "-e"}, null,
    null},
{new String[] {"create", "/zk_test", "my_data", "-c"}, null,
    null},
{new String[] {"create", "/zk_test", "my_data", "-t", "100"},
    null, null},
```

```
{new String[] {"create", "/zk_test", "my_data", "-t", "0"},
    null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-t", "-100"},
    null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-c", "-e"},
    null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-c", "-s"},
    null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-t", "100",
    "-e"}, null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-t", "100",
    "-c"}, null, MalformedCommandException.class},
//added after coverage analysis
{new String[] {"create", "/zk_test", "my_data", "-t", "l"},
    null, MalformedCommandException.class},
{new String[] {"create", "/zk_test", "my_data", "-e", "-s"},
    null, null},
{new String[] {"create", "/zk_test", "my_data",
    "ip:127.0.0.1:crwda"}, null, null},
{new String[] {"create", "/zk_test", "my_data", "-t", "100",
    "-s"}, null, null},
{new String[] {"create", "/zk_test"}, null, null},

{new String[] {"create", "/zk_test", "my_data"}, new
    IllegalArgumentException(), MalformedPathException.class},
{new String[] {"create", "/zk_test", "my_data"}, new
    KeeperException.EphemeralOnLocalSessionException(),
        CliWrapperException.class},
{new String[] {"create", "/zk_test", "my_data"}, new
    KeeperException.InvalidACLException(),
        CliWrapperException.class},
{new String[] {"create", "/zk_test", "my_data"}, new
    InterruptedException(), CliWrapperException.class},
```

Listing 3: Parametri utilizzati ExecTest

Condizioni test

Per fare i test si è utilizzata prevalentemente l'analisi del codice in quanto è risultato difficile trovare documentazione dettagliata riguardanti le classi utilizzate. I progetti sono stati buildati su TravisCI e hanno fatto uso di SonarCloud, questo ha richiesto una modifica del file pom.xml originale, inoltre sono state disabilitate delle funzione non essenziali per riuscire a eseguire il build su macchina locale necessario per l'esecuzione di Pit.

Link Coverage SonarCloud

- Cookie.java https://sonarcloud.io/component_measures?id=cesto93_bookkeeper&metric=Coverage&selected=cesto93_bookkeeper%3Abookkeeper-server%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Fbookkeeper%2Fbookie%2FCookie.java&view=list
- WeightedRandomSelectionImpl.java https://sonarcloud.io/component_measures?id=cesto93_bookkeeper&metric=coverage&selected=cesto93_bookkeeper%3Abookkeeper-server%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Fbookkeeper%2Fclient%2FWeightedRandomSelectionImpl.java
- AvgMinMaxCounter.java https://sonarcloud.io/component_measures?id=cesto93_zookeeper&metric=coverage&selected=cesto93_zookeeper%3Azookeeper-server%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Fzookeeper%2Fserver%2Fmetric%2FAvgMinMaxCounter.java&view=treemap
- CreateCommand.java https://sonarcloud.io/component_measures?id=cesto93_zookeeper&metric=coverage&selected=cesto93_zookeeper%3Azookeeper-server%2Fsrc%2Fmain%2Fjava%2Forg%2Fapache%2Fzookeeper%2Fcli%2FCreateCommand.java&view=treemap