

Report ISW2-Modulo SWTesting

Introduzione

Il seguente report ha come obiettivo quello di presentare l'insieme dei passi svolti e delle problematiche affrontate durante lo svolgimento dell'attività di software testing effettuata su classi Java di due progetti open source Apache, nonché della presentazione e commento dei risultati riportati. Sono stati considerate due classi per il progetto Bookkeeper e tre classi per il progetto AVRO. Le attività di testing e generazione dei risultati, che comprendono metriche di adeguatezza delle test suite, sono state inserite nel ciclo di build dei progetti affinché le operazioni precedentemente descritte fossero effettuate automaticamente ad ogni modifica del progetto o della test suite. A supporto di tale processo è risultato molto utile l'utilizzo di piattaforme come *Travis CI* (per building e testing, usando il paradigma del continuous integration) e *SonarCloud* (per la raccolta dei risultati e la generazione automatica delle metriche di adeguatezza) in associazione con *GitHub*, servizio di version control distribuito. I framework utilizzati sono: *JUnit* per l'implementazione dei casi di test, *JaCoCo* per la generazione dei report relativi al *branch* e *statement coverage* e *PIT* per quanto riguarda la *mutation coverage*.

Scelta delle classi e dei metodi

La scelta delle classi, per quanto riguarda il progetto Apache Bookkeeper, da sottoporre all'attività di test, è stata guidata dall'analisi di alcune metriche in relazione all'ultima versione del progetto.

Una modifica al progetto, relativo alla sezione del professor Falessi, ha permesso un'analisi su tutte le versioni di Bookkeeper, che ha permesso di individuare alcune metriche utili a percepire la propensione di una certa classe, come quelle prese in esame, ad essere affette da problematiche. In particolare, come si può osservare dai risultati riportati in Figura 1, per *WriteCache* e *DigestManager* si nota un elevato valore delle metriche *LOC_added* e *ChgSetSize*. L'aggiunta di un numero elevato di linee di codice aumenta la probabilità di inserire un bug nel codice, mentre il numero elevato di file committati insieme alle classi in esame fanno presumere grandi operazioni di aggiornamento, modifica, aggiunta di funzionalità o risoluzione di problemi che potrebbero far pensare al tralascio di dettagli per i singoli file e quindi introduzione di alterazioni all'interno del codice.

I metodi sono stati scelti in base ad alcune caratteristiche:

- **Valori di ritorno e modificatori:** tutti i metodi scelti presentano valori di ritorno, lanciano eccezioni e sono pubblici o sono testabili interamente fruttando metodi che fanno chiamate a questi, in modo da poter effettuare un controllo diretto sull'esito dell'attività di testing, evitando di controllare parametri particolari di oggetti toccati dall'esecuzione dell'ultimo.
- **Comprensibilità:** sono stati presi in considerazione metodi con una documentazione, ove esistente, altrimenti commentati o di cui fosse intuibile il funzionamento dal codice.
- **Complessità:** legata alla comprensibilità, quest'ultima ha guidato la scelta in modo da evitare metodi troppo complessi da non comprenderne appieno il funzionamento e quindi incorrere in errori nella fase di test.

Implementazione dei casi di test e metriche di adeguatezza

I test sono stati realizzati utilizzando *JUnit* e il runner *Parameterized*, che permette di realizzare test con la stessa struttura semplicemente mediante un metodo statico annotato con *@Parameters* in cui è possibile definire una "matrice" le cui righe rappresentano l'insieme delle invocazioni del costruttore della classe di test e le colonne indicano i parametri da passare ad esso. Inoltre, in alcuni casi sono state utilizzate le annotazioni *@Before* e *@After* che permettono di definire comportamenti preliminari e conclusivi in relazione ad ogni test. Affinché fosse possibile percorrere determinati path all'interno del flusso di esecuzione di un determinato metodo, è stato necessario l'impiego di stub, realizzati attraverso il plugin *Mockito*, per simulare il comportamento di componenti esterni, oppure controllare le chiamate a metodi reali.

Le metriche di adeguatezza per i test scelte sono state *statement coverage*, *branch coverage* e *mutation coverage*.

Bookkeeper

Bookkeeper è un servizio di storage distribuito, scalabile e tollerante ai guasti. Ha una struttura gerarchica costituita da unità fondamentali chiamate *entry* o *record* che costituiscono un log. Un flusso di *entry* costituisce un *ledger* e i server che conservano un insieme di *ledger* sono chiamati *bookie*.

Le classi, appartenenti al modulo Maven *bookkeeper-server*, scelte come oggetto dell'attività dei test sono: WriteCache e DigestManager.

WriteCache

La classe WriteCache, contenuta nel package *org.apache.bookkeeper.bookie.storage.ldb*, si occupa di gestire la cache di scrittura, offre quindi funzionalità di scrittura e lettura sulla cache per le operazioni sul database. Si occupa dell'allocazione della dimensione richiesta e la suddivide in più segmenti. Inoltre, le voci, aggiunte in un buffer, vengono indicizzate tramite hashmap, quindi è possibile scorrerle tramite (*ledgerId*, *entry*), finché la cache non viene flushata.

Il primo metodo testato è *public boolean put(long ledgerId, long entryId, ByteBuf entry)*, il quale permette di scrivere un'entry appartenente ad un certo ledger in cache. A seguito delle attività preliminari alla fase di testing, si sono ottenute le seguenti classi di equivalenza in corrispondenza dei parametri di input:

- ledgerId {< 0, ≥ 0}
- entryId {< 0, ≥ 0}
- entry {valid, invalid, null}

Dal codice si è dedotto che *ledgerId* e *entryId* assumono valori positivi o al più uguali a zero, per cui ci si aspetta che per valori negativi il valore di ritorno del metodo fosse *false*. Inoltre, l'unico modo per rendere *entry* invalid sarebbe consistito nella modifica del parametro *length* della classe *ByteBuf*, ma una lunghezza non valida genera una eccezione al momento della istanziazione dell'oggetto; data l'impossibilità di passare al metodo un buffer non valido, si è escluso questo caso dal test. L'analisi delle classi di equivalenza ha portato alla definizione dei seguenti boundary value:

- ledgerId {-1, 0, 1}
- entryId {-1, 0, 1}
- entry {valid (non vuoto con capacità positiva), null}

Dopo aver eseguito la suite minimale i risultati ottenuti dal plugin JaCoCo hanno evidenziato un livello di *statement* e *branch coverage* rispettivamente del 94% e 50% (Figura 2, Figura 4). Al fine di raggiungere un livello di coverage maggiore, sono stati aggiunti ulteriori casi di test che hanno portato ad un innalzamento delle due metriche di adeguatezza al 98% e 87% (Figura 3, Figura 5). In merito al *mutation testing* si è passato dal *coverage* del 66% (Figura 6), considerando le mutazioni a riga 167 equivalenti rispetto a *strong mutation*, al 83% (Figura 7).

Il secondo metodo considerato è *public ByteBuf get(long ledgerId, long entryId)* che si occupa di recuperare dalla cache la *entry* relativa al *ledgerId* e *entryId*. È stata ottenuta la seguente suddivisione in classi di equivalenza:

- ledgerId {< 0, ≥ 0}
- entryId {< 0, ≥ 0}

Approfondendo lo studio delle chiamate ai metodi effettuate all'interno della *get()*, si è notato che il parametro *ledgerId* non può essere negativo, come nel caso precedentemente discusso.

La *boundary analysis* determina i seguenti valori per i parametri di input:

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

I risultati ottenuti, rispetto a *statement* e *branch coverage*, in seguito all'esecuzione dell'insieme minimale dei casi di test ottenuti a seguito della *category partition*, sono rispettivamente del 95% e 50% (Figura 2, Figura 8). Conseguentemente all'ampliamento della suite di test tali metriche si sono attestate sul 100% (Figura 3, Figura 9) in entrambi i casi. Per quanto concerne il *mutation score*, questo è rimasto invariato, essendo già del 100% (Figura 10).

DigestManager

La classe in questione appartiene al package *org.apache.bookkeeper.proto.checksum* e contiene una serie di metodi che permettono di impacchettare e spaccettare i dati da mandare ad un *Bookie*. Per quanto riguarda il lato mittente, vengono disposte procedure che permettono, considerato un pacchetto di dati, di generare un header di informazioni e computare un digest di integrità. Per quanto riguarda il lato ricevente, invece, viene calcolato il digest sul pacchetto in arrivo e confrontato con quello contenuto in esso.

Tra i metodi presi in esame vi è *public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)*, il quale si occupa di estrarre la *entry* e il *digest* dal pacchetto, eseguire il confronto con quello ricalcolato a partire dai dati del pacchetto e ritornare il buffer che costituisce la *entry* stessa. Segue la suddivisione in classi di equivalenza:

- *entryId* {< 0, ≥ 0}
- *dataReceived* {valid, invalid, null}

Per *dataReceived* il caso *invalid* non può essere definito dato che qualsiasi buffer è ammesso meno che uno di dimensione < 0 che risulta impossibile da instanziare. I valori di boundary risultano:

- *entryId* {-1, 0, 1}
- *dataReceived* {Unpooled.buffer con lunghezza > 0, null}

La suite minimale riporta *statement e branch coverage* del 100% (Figura 11, Figura 13) e *mutation score* del 66% (Figura 14), che sale al 100% (Figura 15) a seguito dell'estensione dei casi di test.

Un metodo strettamente collegato al precedente è *private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryCheck)* che ne implementa quasi la totalità della logica, infatti viene testato attraverso chiamate a quest'ultimo, essendo pubblico. Dal codice si è evinto che il parametro *skipEntryCheck* è fissato al valore *false*. Di seguito i risultati della *category partition*:

- *entryId* {< 0, ≥ 0}
- *dataReceived* {valid, invalid, null}
- *skipEntryCheck* {true, false}

I valori di boundary risultano:

- *entryId* {-1, 0, 1}
- *dataReceived* {Unpooled.buffer con lunghezza > 0, null}
- *skipEntryCheck* {false}

La suite minimale riporta *statement e branch coverage* rispettivamente del 48% e 50% (Figura 11, Figura 16) e *mutation score* del 58% (Figura 18), che salgono rispettivamente al 100%, 90% (Figura 12, Figura 17) e 100% (Figura 19), a seguito dell'estensione dei casi di test. Affinché fosse possibile innalzare il livello di coverage è stato necessario definire un buffer con un header considerato errato, ovvero bypassare la corretta costruzione del pacchetto attraverso metodi come *computeDigestAndPackageForSending()*.

Come terzo metodo si è scelto *public long verifyDigestAndReturnLac(ByteBuf dataReceived)*, che permette di controllare l'integrità del messaggio e ricavare il LAC, ovvero l'ID dell'ultima *entry* confermata, ovvero registrata da Bookkeeper con successo.

Segue la suddivisione in classi di equivalenza:

- *dataReceived* {valid, invalid, null}

Per *dataReceived* il caso *invalid* non può essere definito dato che qualsiasi buffer è ammesso meno che uno di dimensione < 0 che risulta impossibile da instanziare. I valori di boundary risultano:

- *dataReceived* {Unpooled.buffer con lunghezza > 0, null}

La suite minimale riporta *statement e branch coverage* rispettivamente del 49% e 50% (Figura 11, Figura 20) e *mutation score* del 88% (Figura 22), che salgono tutte al 100% (Figura 12, Figura 21, Figura 23), a seguito dell'estensione dei casi di test.

L'ultimo metodo analizzato è *public static byte[] generateMasterKey(byte[] password)*, il quale genera un'array di byte che contengono il digest estratto dalla *password* e che rappresenta una chiave riferita al ledger.

Di seguito i risultati della *category partition*:

- *password* {valid, void}

I valori di boundary risultano:

- *password* {random byte array, ""}

La suite minimale riporta *statement* e *branch coverage* del 100% (Figura 11, Figura 24) e *mutation score* del 100% (Figura 25), poiché la mutazione sopravvissuta risulta equivalente al SUT rispetto a strong mutation.

AVRO

Avro è un servizio di serializzazione di dati che fornisce un formato binario compatto e veloce. Si basa sull'idea di *schema*, un modello che permette serializzazione e deserializzazione del dato senza overhead. Gli schemi possono essere customizzati e descritti usando il formato JSON. Supporta le RPC e lo schema utilizzato nella comunicazione tra client e server viene deciso durante l'handshake.

Le classi analizzate, appartenenti al modulo Maven *lang/java/avro*, sono: *BinaryData*, *SpecificData* e *GenericData*.

BinaryData

Questa classe, contenuta nel package *org.apache.avro.io*, fornisce strumenti per codificare in binario i dati.

Il primo metodo che si è deciso di testare è *public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)*. Da documentazione, tale metodo permette di comparare lessicograficamente due array di byte e restituire un valore positivo se il primo è maggiore del secondo, un valore negativo se accade il viceversa e zero se risultano uguali. I parametri *b* rappresentano gli array di byte da comparare, i parametri *s* sono le posizioni da cui cominciare la comparazione e i parametri *l* indicano la porzione di array da comparare a partire da *s*. È stata effettuata la *Category Partition* unidimensionale sui parametri di input. Le scelte dei valori delle classi di equivalenza e sull'analisi dei boundary-value sono state guidate dalla documentazione e quindi dal significato di ogni singolo parametro. Le classi di equivalenza identificate sono le seguenti:

- *b1* {valid, void} ¹
- *b2* {valid, void} ¹
- *s1* {0, >0} ²
- *l1* {0, >0} ²
- *s2* {0, >0} ²
- *l2* {0, >0} ²

Sono stati individuati i boundary-value approfondendo l'analisi delle classi di equivalenza come segue:

- *b1* {"test1".getBytes(), "".getBytes()}
- *b2* {"test2".getBytes(), "".getBytes()}
- *s1* {0, 1}
- *l1* {0, 4}
- *s2* {0, 1}
- *l2* {0, 4}

In seguito all'esecuzione della suite minimale i test evidenziano un corretto funzionamento a livello semantico rispetto alla documentazione fornita, inoltre lo *statement* e *branch coverage* sono rispettivamente del 100% e 83% (Figura 26, Figura 28). Si è reso necessario l'ampliamento della test suite per aumentare il *branch coverage*, attestandosi così sul 100% (Figura 27, Figura 29). Mentre per quanto riguarda la *mutation coverage*, in seguito all'esecuzione della suite estesa, è del 87% (Figura 30), mentre risulta del 100% (Figura 31) in seguito all'ulteriore ampliamento introducendo casi di test per uccidere mutazioni relative alla modifica dei valori di boundary nelle condizioni di arresto del ciclo for e della sostituzione del meno con il più nel valore di ritorno.

Il secondo metodo testato è *public static int encodeInt(int n, byte[] buf, int pos)*. Da documentazione tale metodo restituisce il numero dei byte scritti nel buffer *buf* che contiene la codifica dell'intero *n*, inoltre si raccomanda una dimensione minima di cinque byte per quanto riguarda *buf*. Essendo *buf* un array che già contiene dati e di lunghezza maggiore rispetto alla codifica di *n*, *pos* può assumere valori negativi e sovrascrivere porzioni di array. È stata effettuata *Category Partition unidimensionale* sui parametri di input come segue:

- *n* {< 0, ≥ 0}

¹ il caso relativo alla lunghezza massima non è stato analizzato poiché non vengono imposti vincoli sulla dimensione dell'array

² Il caso relativo a valori negativi non è stato analizzato in quanto si tratta di posizioni iniziali o lunghezze che per loro natura non possono assumere tali valori

- buf {valid, void}
- pos {< 0, ≥ 0}

Approfondendo l'analisi delle classi di equivalenza sono stati individuati i seguenti boundary-value:

- n {-1, 0, 1}
- buf {new byte[5], new byte[0]}
- pos {-1, 0, 1}

In seguito all'esecuzione della suite minimale i test evidenziano un corretto funzionamento a livello semantico rispetto alla documentazione fornita, inoltre lo *statement e branch coverage* sono rispettivamente del 26% e 12% (Figura 26, Figura 32). Si è reso necessario l'ampliamento della test suite per aumentare *statement e branch coverage* attestandosi così sul 100% (Figura 27, Figura 33) in entrambi i casi. Mentre per quanto riguarda la *mutation coverage*, in seguito all'esecuzione della suite estesa, è del 44% (Figura 34), mentre risulta del 100% (Figura 35) in seguito all'ulteriore ampliamento della test suite introducendo il controllo sul *buf* attraverso l'hash, questo ha permesso di uccidere mutazioni sulla codifica relative a sostituzioni di AND e OR. Tali mutazioni non sono state rilevate nella suite precedente poiché il controllo è stato effettuato soltanto sul valore di ritorno. Il *mutation score* è salito al 100%, considerando che le mutazioni alle linee 323, 326 e 329 sono equivalenti al SUT rispetto a strong e weak mutation, poiché soltanto con un *n* dispari sarebbe possibile eliminare la mutazione sul valore di boundary dell'if, ma per via dello statement a linea 318 *n* assume un valore pari poiché moltiplicato per due.

SpecificData

La classe, contenuta nel package *org.apache.avro.specific*, offre degli strumenti per la generazione e la gestione di oggetti di tipo schema per tipi di dato specifici del linguaggio Java. Si è scelto di testare il metodo *public Class getClass(Schema schema)*. Il metodo restituisce il tipo di classe che implementa lo schema, oppure *null* se non esiste. La classe *Schema* è una classe astratta che fornisce la sua reale istanziazione attraverso costruttori che dipendono dal tipo di schema *Schema.Type*. Per questa motivazione la *Category Partition* è definita secondo la considerazione dell'intera enumerazione *Schema.Type*, come segue:

- schema {null, Schema.Type.INT, Schema.Type.BOOLEAN, Schema.Type.NULL, Schema.Type.LONG, Schema.Type.FLOAT, Schema.Type.DOUBLE, Schema.Type.BYTES, Schema.Type.STRING, Schema.Type.MAP, Schema.Type.ARRAY, Schema.Type.RECORD, Schema.Type.FIXED, Schema.Type.ENUM, Schema.Type.UNION}

Poiché si tratta di una enumerazione la *Category Partition* coincide con i boundary-value.

A seguito dell'esecuzione della suite minimale i test evidenziano un corretto funzionamento a livello semantico rispetto alla documentazione fornita, inoltre lo *statement e branch coverage* sono rispettivamente del 36% e 17% (Figura 36, Figura 38). Si è reso necessario l'ampliamento della test suite per aumentare *statement e branch coverage* attestandosi così sul 87% e 96% (Figura 37, Figura 39). Il risultato del *mutation test* si attesta sul 100% (Figura 40), considerando che la mutazione relativa alla linea 253 è equivalente al SUT rispetto a *strong mutation*, ma non rispetto a *weak mutation*, poiché restituisce lo stesso valore, ma non segue lo stesso path di esecuzione. Affinché fosse possibile l'incremento dello score rispetto a *branch e statement coverage* è stato necessaria l'introduzione della classe *OuterClassForTest*, che contiene la classe innestata *InnerClassForTest*. Inoltre, per il medesimo motivo, è stato necessario l'impiego di *Mockito* ed in particolare il metodo *spy*, che permette di istanziare normalmente un oggetto, ma è possibile intercettare e modificare il comportamento di determinati metodi o attributi.

GenericData

La classe in questione appartiene al package *org.apache.avro.generic* e offre un set di utility per dati Java generici. Il metodo testato è *private Object deepCopyRaw(Schema schema, Object value)*. Tale metodo è stato testato attraverso chiamate al metodo pubblico *public <T> T deepCopy(Schema schema, T value)*, che comprende una serie di operazioni preliminari alla chiamata del metodo testato. Il metodo ha l'obiettivo di creare una copia di un determinato oggetto dato il suo schema. Si è notato che con i tipi di dato che si riferiscono a schemi di tipo *BOOLEAN, DOUBLE, FLOAT, INT, LONG* non vi è relazione con il tipo di dato che rappresenta *value*, infatti viene semplicemente ritornata la copia, poiché si tratta di tipi di dato primitivi.

Come nel caso precedente la classe *Schema* è una classe astratta che fornisce la sua reale istanziazione attraverso costruttori che dipendono dal tipo di schema *Schema.Type*. Per questa motivazione la *Category Partition* è definita secondo la considerazione dell'intera enumerazione *Schema.Type*, come segue:

- schema {null, *Schema.Type.INT*, *Schema.Type.BOOLEAN*, *Schema.Type.NULL*, *Schema.Type.LONG*, *Schema.Type.FLOAT*, *Schema.Type.DOUBLE*, *Schema.Type.BYTES*, *Schema.Type.STRING*, *Schema.Type.MAP*, *Schema.Type.ARRAY*, *Schema.Type.RECORD*, *Schema.Type.FIXED*, *Schema.Type.ENUM*, *Schema.Type.UNION*}
- value {valid, null, invalid}

Poiché si tratta di una enumerazione la *Category Partition* coincide con i boundary-value, e *value* è per lo più del tipo relativo allo schema, tranne nel caso *invalid*. La suite minimale fornisce il massimo livello di *branch* e *statement coverage* che si attesta rispettivamente sul 90% e 91% (Figura 41, Figura 42). Non considerando il caso di default dello switch, il quale risulta irraggiungibile poiché esiste un case per ogni valore della enumerazione *Schema.Type*, e le linee 1236 e 1237, che vengono coperte alle linee 1220 e 1221, il *coverage* si attesta sul 100% in entrambi i casi. Per quanto riguarda il *mutation score* questo risulta del 94% (Figura 43) con l'esecuzione della suite minimale, mentre si attesta sul 100% (Figura 44) con l'ampliamento dei casi di test. Affinchè fosse raggiunto il massimo valore di *mutation coverage*, è stato impiegato il metodo *verify* di *Mockito* che permette di monitorare l'effettiva chiamata di un metodo relativo all'oggetto in questione; in questo modo è stato possibile uccidere la mutazione alla linea 1287.

Version	File Name	LOC added	MAX LOC added	AVG LOC added	Churn	MAX Churn	AVG Churn	NR	ChgSetSize	MAX ChgSet	AVG ChgSet	Buggy
14	WriteCache.java	7886	303	148,7924528	306	303	5,773584906	53	61880	1650	1167,54717	NO
14	DigestManager.java	4390	254	107,0731707	21	254	0,512195122	41	45792	1650	1116,878049	NO

Figura 1 Metriche Bookkeeper

Apache BookKeeper :: Tests > bookkeeper-server > org.apache.bookkeeper.bookie.storage.Ldb > WriteCache

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)	<div></div>	0%	<div></div>	0%	8	8	32	32	1	1
lambda\$forEach\$0(long, long, long, long)	<div></div>	0%	<div></div>	0%	2	2	8	8	1	1
getLastEntry(long)	<div></div>	0%	<div></div>	0%	2	2	4	4	1	1
isEmpty()	<div></div>	0%	<div></div>	0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)	<div></div>	0%		n/a	1	1	2	2	1	1
deleteLedger(long)	<div></div>	0%		n/a	1	1	2	2	1	1
put(long, long, ByteBuf)	<div></div>	94%	<div></div>	50%	4	5	4	20	0	1
size()	<div></div>	0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)	<div></div>	98%	<div></div>	66%	2	4	0	25	0	1
get(long, long)	<div></div>	95%	<div></div>	50%	1	2	1	10	0	1
clear()	<div></div>	100%		n/a	0	1	0	7	0	1
close()	<div></div>	100%	<div></div>	100%	0	2	0	3	0	1
alignToPowerOfTwo(long)	<div></div>	100%		n/a	0	1	0	1	0	1
static {...}	<div></div>	100%		n/a	0	1	0	2	0	1
align64(int)	<div></div>	100%		n/a	0	1	0	1	0	1
count()	<div></div>	100%		n/a	0	1	0	1	0	1
Total	271 of 617	56%	27 of 38	28%	24	35	55	120	7	16

Figura 2 Statement e branch coverage suite minimale WriteCache

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda\$forEach\$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)		0%		n/a	1	1	2	2	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		98%		87%	1	5	1	20	0	1
get(long, long)		100%		100%	0	2	0	10	0	1
clear()		100%		n/a	0	1	0	7	0	1
close()		100%		100%	0	2	0	3	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
count()		100%		n/a	0	1	0	1	0	1
Total	265 of 617	57%	23 of 38	39%	20	35	51	120	7	16

Figura 3 Statement e branch coverage suite estesa WriteCache

```

131. public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132.     int size = entry.readableBytes();
133.
134.     // Align to 64 bytes so that different threads will not contend the same L1
135.     // cache line
136.     int alignedSize = align64(size);
137.
138.     long offset;
139.     int localOffset;
140.     int segmentIdx;
141.
142.     while (true) {
143.         offset = cacheOffset.getAndAdd(alignedSize);
144.         localOffset = (int) (offset & segmentOffsetMask);
145.         segmentIdx = (int) (offset >>> segmentOffsetBits);
146.
147.         if ((offset + size) > maxCacheSize) {
148.             // Cache is full
149.             return false;
150.         } else if (maxSegmentSize - localOffset < size) {
151.             // If an entry is at the end of a segment, we need to get a new offset and try
152.             // again in next segment
153.             continue;
154.         } else {
155.             // Found a good offset
156.             break;
157.         }
158.     }
159.
160.     cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161.
162.     // Update last entryId for ledger. This logic is to handle writes for the same
163.     // ledger coming out of order and from different thread, though in practice it
164.     // should not happen and the compareAndSet should be always uncontended.
165.     while (true) {
166.         long currentLastEntryId = lastEntryMap.get(ledgerId);
167.         if (currentLastEntryId > entryId) {
168.             // A newer entry is already there
169.             break;
170.         }
171.
172.         if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173.             break;
174.         }
175.     }
176.
177.     index.put(ledgerId, entryId, offset, size);
178.     cacheCount.increment();
179.     cacheSize.addAndGet(size);
180.     return true;
181. }

```

Figura 4 Statement e branch coverage suite minimale put()

```

130.
131. public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132.     int size = entry.readableBytes();
133.
134.     // Align to 64 bytes so that different threads will not contend the same L1
135.     // cache line
136.     int alignedSize = align64(size);
137.
138.     long offset;
139.     int localOffset;
140.     int segmentIdx;
141.
142.     while (true) {
143.         offset = cacheOffset.getAndAdd(alignedSize);
144.         localOffset = (int) (offset & segmentOffsetMask);
145.         segmentIdx = (int) (offset >>> segmentOffsetBits);
146.
147.         if ((offset + size) > maxCacheSize) {
148.             // Cache is full
149.             return false;
150.         } else if (maxSegmentSize - localOffset < size) {
151.             // If an entry is at the end of a segment, we need to get a new offset and try
152.             // again in next segment
153.             continue;
154.         } else {
155.             // Found a good offset
156.             break;
157.         }
158.     }
159.
160.     cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161.
162.     // Update last entryId for ledger. This logic is to handle writes for the same
163.     // ledger coming out of order and from different thread, though in practice it
164.     // should not happen and the compareAndSet should be always uncontended.
165.     while (true) {
166.         long currentLastEntryId = lastEntryMap.get(ledgerId);
167.         if (currentLastEntryId > entryId) {
168.             // A newer entry is already there
169.             break;
170.         }
171.
172.         if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173.             break;
174.         }
175.     }
176.
177.     index.put(ledgerId, entryId, offset, size);
178.     cacheCount.increment();
179.     cacheSize.addAndGet(size);
180.     return true;
181. }

```

Figura 5 Statement e branch coverage suite estesa put()


```

131 public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132     int size = entry.readableBytes();
133
134     // Align to 64 bytes so that different threads will not contend the same L1
135     // cache line
136     int alignedSize = align64(size);
137
138     long offset;
139     int localOffset;
140     int segmentIdx;
141
142     while (true) {
143         offset = cacheOffset.getAndAdd(alignedSize);
144         localOffset = (int) (offset & segmentOffsetMask);
145         segmentIdx = (int) (offset >>> segmentOffsetBits);
146
147         if ((offset + size) > maxCacheSize) {
148             // Cache is full
149             return false;
150         } else if (maxSegmentSize - localOffset < size) {
151             // If an entry is at the end of a segment, we need to get a new offset and try
152             // again in next segment
153             continue;
154         } else {
155             // Found a good offset
156             break;
157         }
158     }
159
160     cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161
162     // Update last entryId for ledger. This logic is to handle writes for the same
163     // ledger coming out of order and from different thread, though in practice it
164     // should not happen and the compareAndSet should be always uncontended.
165     while (true) {
166         long currentLastEntryId = lastEntryMap.get(ledgerId);
167         if (currentLastEntryId > entryId) {
168             // A newer entry is already there
169             break;
170         }
171
172         if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173             break;
174         }
175     }
176
177     index.put(ledgerId, entryId, offset, size);
178     cacheCount.increment();
179     cacheSize.addAndGet(size);
180     return true;
181 }

```

Figura 6 Mutation coverage suite minimale put()

```

131 public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132     int size = entry.readableBytes();
133
134     // Align to 64 bytes so that different threads will not contend the same L1
135     // cache line
136     int alignedSize = align64(size);
137
138     long offset;
139     int localOffset;
140     int segmentIdx;
141
142     while (true) {
143         offset = cacheOffset.getAndAdd(alignedSize);
144         localOffset = (int) (offset & segmentOffsetMask);
145         segmentIdx = (int) (offset >>> segmentOffsetBits);
146
147         if ((offset + size) > maxCacheSize) {
148             // Cache is full
149             return false;
150         } else if (maxSegmentSize - localOffset < size) {
151             // If an entry is at the end of a segment, we need to get a new offset and try
152             // again in next segment
153             continue;
154         } else {
155             // Found a good offset
156             break;
157         }
158     }
159
160     cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161
162     // Update last entryId for ledger. This logic is to handle writes for the same
163     // ledger coming out of order and from different thread, though in practice it
164     // should not happen and the compareAndSet should be always uncontended.
165     while (true) {
166         long currentLastEntryId = lastEntryMap.get(ledgerId);
167         if (currentLastEntryId > entryId) {
168             // A newer entry is already there
169             break;
170         }
171
172         if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173             break;
174         }
175     }
176
177     index.put(ledgerId, entryId, offset, size);
178     cacheCount.increment();
179     cacheSize.addAndGet(size);
180     return true;
181 }

```

Figura 7 Mutation coverage suite estesa put()

```

183.     public ByteBuf get(long ledgerId, long entryId) {
184.         LongPair result = index.get(ledgerId, entryId);
185.         if (result == null) {
186.             return null;
187.         }
188.
189.         long offset = result.first;
190.         int size = (int) result.second;
191.         ByteBuf entry = allocator.buffer(size, size);
192.
193.         int localOffset = (int) (offset & segmentOffsetMask);
194.         int segmentIdx = (int) (offset >>> segmentOffsetBits);
195.         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196.         return entry;
197.     }

```

Figura 8 Statement e branch coverage suite minimale get()

```

182.
183.     public ByteBuf get(long ledgerId, long entryId) {
184.         LongPair result = index.get(ledgerId, entryId);
185.         if (result == null) {
186.             return null;
187.         }
188.
189.         long offset = result.first;
190.         int size = (int) result.second;
191.         ByteBuf entry = allocator.buffer(size, size);
192.
193.         int localOffset = (int) (offset & segmentOffsetMask);
194.         int segmentIdx = (int) (offset >>> segmentOffsetBits);
195.         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196.         return entry;
197.     }

```

Figura 9 Statement e branch coverage suite estesa get()

```

183     public ByteBuf get(long ledgerId, long entryId) {
184         LongPair result = index.get(ledgerId, entryId);
185         if (result == null) {
186             return null;
187         }
188
189         long offset = result.first;
190         int size = (int) result.second;
191         ByteBuf entry = allocator.buffer(size, size);
192
193         int localOffset = (int) (offset & segmentOffsetMask);
194         int segmentIdx = (int) (offset >>> segmentOffsetBits);
195         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196         return entry;
197     }

```

Figura 10 Mutation coverage get()

Apache BookKeeper :: Tests > bookkeeper-server > org.apache.bookkeeper.proto.checksum > DigestManager

DigestManager

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
verifyDigest(long, ByteBuf, boolean)	<div><div></div></div>	48%	<div><div></div></div>	50%	5 6	9 22	0 1
verifyDigestAndReturnLac(ByteBuf)	<div><div></div></div>	49%	<div><div></div></div>	50%	3 4	7 17	0 1
instantiate(long, byte[], DataFormats.LedgerMetadataFormat.DigestType, ByteBufAllocator, boolean)	<div><div></div></div>	43%	<div><div></div></div>	40%	3 5	3 6	0 1
verifyDigestAndReturnLastConfirmed(ByteBuf)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	6 6	1 1
computeDigestAndPackageForSending(long, long, long, ByteBuf)	<div><div></div></div>	81%	<div><div></div></div>	50%	1 2	1 11	0 1
update(byte[])	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	2 2	1 1
computeDigestAndPackageForSendingLac(long)	<div><div></div></div>	83%	<div><div></div></div>	50%	1 2	1 8	0 1
verifyDigest(ByteBuf)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1 1	2 2	1 1
DigestManager(long, boolean, ByteBufAllocator)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 6	0 1
verifyDigestAndReturnData(long, ByteBuf)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 3	0 1
generateMasterKey(byte[])	<div><div></div></div>	100%	<div><div></div></div>	100%	0 2	0 1	0 1
verifyDigest(long, ByteBuf)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 2	0 1
static (...)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0 1	0 1	0 1
Total	193 of 441	56%	13 of 27	51%	16 28	31 87	3 13

Figura 11 Statement e branch coverage suite minimale DigestManager

DigestManager

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
verifyDigestAndReturnLastConfirmed(ByteBuf)		0%		n/a	1	1	6	6	1	1
Instantiate(long, byte[], DataFormats.LedgerMetadataFormat.DigestType, ByteBufAllocator, boolean)		73%		80%	1	5	1	6	0	1
computeDigestAndPackageForSending(long, long, long, ByteBuf)		81%		50%	1	2	1	11	0	1
update(byte[])		0%		n/a	1	1	2	2	1	1
verifyDigest(ByteBuf)		0%		n/a	1	1	2	2	1	1
verifyDigest(long, ByteBuf, boolean)		100%		90%	1	6	0	22	0	1
verifyDigestAndReturnLac(ByteBuf)		100%		100%	0	4	0	17	0	1
computeDigestAndPackageForSendingLac(long)		100%		100%	0	2	0	8	0	1
DigestManager(long, boolean, ByteBufAllocator)		100%		n/a	0	1	0	6	0	1
verifyDigestAndReturnData(long, ByteBuf)		100%		n/a	0	1	0	3	0	1
generateMasterKey(byte[])		100%		100%	0	2	0	1	0	1
verifyDigest(long, ByteBuf)		100%		n/a	0	1	0	2	0	1
static(...)		100%		n/a	0	1	0	1	0	1
Total	57 of 441	87%	3 of 27	88%	6	28	12	87	3	13

Figura 12 Statement e branch coverage suite estesa DigestManager

```

227.    /**
228.     * Verify that the digest matches and returns the data in the entry.
229.     *
230.     * @param entryId
231.     * @param dataReceived
232.     * @return
233.     * @throws BKDigestMatchException
234.     */
235.    public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
236.        throws BKDigestMatchException {
237.        verifyDigest(entryId, dataReceived);
238.        dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
239.        return dataReceived;
240.    }

```

Figura 13 Statement e branch coverage verifyDigestAndReturnData()

```

227    /**
228     * Verify that the digest matches and returns the data in the entry.
229     *
230     * @param entryId
231     * @param dataReceived
232     * @return
233     * @throws BKDigestMatchException
234     */
235    public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
236        throws BKDigestMatchException {
237.1    verifyDigest(entryId, dataReceived);
238.1    dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
239.1    return dataReceived;
240    }

```

Figura 14 Mutation coverage suite minimale verifyDigestAndReturnData()

```

227    /**
228     * Verify that the digest matches and returns the data in the entry.
229     *
230     * @param entryId
231     * @param dataReceived
232     * @return
233     * @throws BKDigestMatchException
234     */
235    public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
236        throws BKDigestMatchException {
237.1    verifyDigest(entryId, dataReceived);
238.1    dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
239.1    return dataReceived;
240    }

```

Figura 15 Mutation coverage suite estesa verifyDigestAndReturnData()

```

150.     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
151.         throws BKDigestMatchException {
152.
153.         if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
154.             logger.error("Data received is smaller than the minimum for this digest type. "
155.                 + " Either the packet is corrupt, or the wrong digest is configured. "
156.                 + " Digest type: {}, Packet Length: {}",
157.                 this.getClass().getName(), dataReceived.readableBytes());
158.             throw new BKDigestMatchException();
159.         }
160.         update(dataReceived.slice(0, METADATA_LENGTH));
161.
162.         int offset = METADATA_LENGTH + macCodeLength;
163.         update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
164.
165.         ByteBuf digest = allocator.buffer(macCodeLength);
166.         populateValueAndReset(digest);
167.
168.         try {
169.             if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
170.                 logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
171.                 throw new BKDigestMatchException();
172.             }
173.         } finally {
174.             digest.release();
175.         }
176.
177.         long actualLedgerId = dataReceived.readLong();
178.         long actualEntryId = dataReceived.readLong();
179.
180.         if (actualLedgerId != ledgerId) {
181.             logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
182.                 + actualLedgerId);
183.             throw new BKDigestMatchException();
184.         }
185.
186.         if (!skipEntryIdCheck && actualEntryId != entryId) {
187.             logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
188.                 + actualEntryId);
189.             throw new BKDigestMatchException();
190.         }
191.     }
192. }

```

Figura 16 Statement e branch coverage suite minmale verifyDigest()

```

149.     private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
150.         throws BKDigestMatchException {
151.
152.         if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
153.             logger.error("Data received is smaller than the minimum for this digest type. "
154.                 + " Either the packet is corrupt, or the wrong digest is configured. "
155.                 + " Digest type: {}, Packet Length: {}",
156.                 this.getClass().getName(), dataReceived.readableBytes());
157.             throw new BKDigestMatchException();
158.         }
159.         update(dataReceived.slice(0, METADATA_LENGTH));
160.
161.         int offset = METADATA_LENGTH + macCodeLength;
162.         update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
163.
164.         ByteBuf digest = allocator.buffer(macCodeLength);
165.         populateValueAndReset(digest);
166.
167.         try {
168.             if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
169.                 logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
170.                 throw new BKDigestMatchException();
171.             }
172.         } finally {
173.             digest.release();
174.         }
175.
176.         long actualLedgerId = dataReceived.readLong();
177.         long actualEntryId = dataReceived.readLong();
178.
179.         if (actualLedgerId != ledgerId) {
180.             logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
181.                 + actualLedgerId);
182.             throw new BKDigestMatchException();
183.         }
184.
185.         if (!skipEntryIdCheck && actualEntryId != entryId) {
186.             logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
187.                 + actualEntryId);
188.             throw new BKDigestMatchException();
189.         }
190.     }
191. }
192. }

```

Figura 17 Statement e branch coverage suite estesa verifyDigest()

```

150 private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
151     throws BKDigestMatchException {
152
153 2 if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
154     logger.error("Data received is smaller than the minimum for this digest type. "
155         + " Either the packet it corrupt, or the wrong digest is configured. "
156         + " Digest type: {}, Packet Length: {}",
157         this.getClass().getName(), dataReceived.readableBytes());
158     throw new BKDigestMatchException();
159 }
160 1 update(dataReceived.slice(0, METADATA_LENGTH));
161
162 1 int offset = METADATA_LENGTH + macCodeLength;
163 2 update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
164
165     ByteBuf digest = allocator.buffer(macCodeLength);
166 1 populateValueAndReset(digest);
167
168     try {
169 1 if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
170     logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
171     throw new BKDigestMatchException();
172 }
173 } finally {
174     digest.release();
175 }
176
177     long actualLedgerId = dataReceived.readLong();
178     long actualEntryId = dataReceived.readLong();
179
180 1 if (actualLedgerId != ledgerId) {
181     logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
182         + actualLedgerId);
183     throw new BKDigestMatchException();
184 }
185
186 2 if (!skipEntryIdCheck && actualEntryId != entryId) {
187     logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
188         + actualEntryId);
189     throw new BKDigestMatchException();
190 }
191
192 }

```

Figura 18 Mutation coverage suite minmale verifyDigest()

```

150 private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
151     throws BKDigestMatchException {
152
153 3 if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
154     logger.error("Data received is smaller than the minimum for this digest type. "
155         + " Either the packet it corrupt, or the wrong digest is configured. "
156         + " Digest type: {}, Packet Length: {}",
157         this.getClass().getName(), dataReceived.readableBytes());
158     throw new BKDigestMatchException();
159 }
160 1 update(dataReceived.slice(0, METADATA_LENGTH));
161
162 1 int offset = METADATA_LENGTH + macCodeLength;
163 2 update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
164
165     ByteBuf digest = allocator.buffer(macCodeLength);
166 1 populateValueAndReset(digest);
167
168     try {
169 1 if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
170     logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
171     throw new BKDigestMatchException();
172 }
173 } finally {
174     digest.release();
175 }
176
177     long actualLedgerId = dataReceived.readLong();
178     long actualEntryId = dataReceived.readLong();
179
180 1 if (actualLedgerId != ledgerId) {
181     logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + ", actual: "
182         + actualLedgerId);
183     throw new BKDigestMatchException();
184 }
185
186 2 if (!skipEntryIdCheck && actualEntryId != entryId) {
187     logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + ", actual: "
188         + actualEntryId);
189     throw new BKDigestMatchException();
190 }
191
192 }

```

Figura 19 Mutation coverage suite estesa verifyDigest()


```

194. public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
195.     ◆ if ((LAC METADATA LENGTH + macCodeLength) > dataReceived.readableBytes()) {
196.         logger.error("Data received is smaller than the minimum for this digest type."
197.             + " Either the packet it corrupt, or the wrong digest is configured. "
198.             + " Digest type: {}, Packet Length: {}",
199.             this.getClass().getName(), dataReceived.readableBytes());
200.         throw new BKDigestMatchException();
201.     }
202.
203.     update(dataReceived.slice(0, LAC_METADATA_LENGTH));
204.
205.     ByteBuf digest = allocator.buffer(macCodeLength);
206.     try {
207.         populateValueAndReset(digest);
208.
209.         ◆ if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
210.             logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
211.             throw new BKDigestMatchException();
212.         }
213.     } finally {
214.         digest.release();
215.     }
216.
217.     long actualLedgerId = dataReceived.readLong();
218.     long lac = dataReceived.readLong();
219.     ◆ if (actualLedgerId != ledgerId) {
220.         logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
221.             + actualLedgerId);
222.         throw new BKDigestMatchException();
223.     }
224.     return lac;
225. }

```

Figura 20 Statement e branch coverage suite minmale verifyDigestAndReturnLac()

```

193. public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
194.     ◆ if ((LAC METADATA LENGTH + macCodeLength) > dataReceived.readableBytes()) {
195.         logger.error("Data received is smaller than the minimum for this digest type."
196.             + " Either the packet it corrupt, or the wrong digest is configured. "
197.             + " Digest type: {}, Packet Length: {}",
198.             this.getClass().getName(), dataReceived.readableBytes());
199.         throw new BKDigestMatchException();
200.     }
201.
202.     update(dataReceived.slice(0, LAC_METADATA_LENGTH));
203.
204.     ByteBuf digest = allocator.buffer(macCodeLength);
205.     try {
206.         populateValueAndReset(digest);
207.
208.         ◆ if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
209.             logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
210.             throw new BKDigestMatchException();
211.         }
212.     } finally {
213.         digest.release();
214.     }
215.
216.     long actualLedgerId = dataReceived.readLong();
217.     long lac = dataReceived.readLong();
218.     ◆ if (actualLedgerId != ledgerId) {
219.         logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
220.             + actualLedgerId);
221.         throw new BKDigestMatchException();
222.     }
223.     return lac;
224. }
225.
226.

```

Figura 21 Statement e branch coverage suite estesa verifyDigestAndReturnLac()

```

194 public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
195 3 if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
196     logger.error("Data received is smaller than the minimum for this digest type."
197         + " Either the packet is corrupt, or the wrong digest is configured. "
198         + " Digest type: {}, Packet Length: {}",
199         this.getClass().getName(), dataReceived.readableBytes());
200     throw new BKDigestMatchException();
201 }
202
203 1 update(dataReceived.slice(0, LAC_METADATA_LENGTH));
204
205     ByteBuf digest = allocator.buffer(macCodeLength);
206     try {
207 1 populateValueAndReset(digest);
208
209 1 if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
210     logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
211     throw new BKDigestMatchException();
212 }
213 } finally {
214     digest.release();
215 }
216
217     long actualLedgerId = dataReceived.readLong();
218     long lac = dataReceived.readLong();
219 1 if (actualLedgerId != ledgerId) {
220     logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
221         + actualLedgerId);
222     throw new BKDigestMatchException();
223 }
224 1 return lac;
225 }

```

Figura 22 Mutation coverage suite minimale verifyDigestAndReturnLac()

```

194 public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
195 3 if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
196     logger.error("Data received is smaller than the minimum for this digest type."
197         + " Either the packet is corrupt, or the wrong digest is configured. "
198         + " Digest type: {}, Packet Length: {}",
199         this.getClass().getName(), dataReceived.readableBytes());
200     throw new BKDigestMatchException();
201 }
202
203 1 update(dataReceived.slice(0, LAC_METADATA_LENGTH));
204
205     ByteBuf digest = allocator.buffer(macCodeLength);
206     try {
207 1 populateValueAndReset(digest);
208
209 1 if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
210     logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
211     throw new BKDigestMatchException();
212 }
213 } finally {
214     digest.release();
215 }
216
217     long actualLedgerId = dataReceived.readLong();
218     long lac = dataReceived.readLong();
219 1 if (actualLedgerId != ledgerId) {
220     logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
221         + actualLedgerId);
222     throw new BKDigestMatchException();
223 }
224 1 return lac;
225 }

```

Figura 23 Mutation coverage suite estesa verifyDigestAndReturnLac()

```

86. public static byte[] generateMasterKey(byte[] password) throws NoSuchAlgorithmException {
87. 3 return password.length > 0 ? MacDigestManager.genDigest("ledger", password) : MacDigestManager.EMPTY_LEDGER_KEY;
88. }

```

Figura 24 Statement e branch coverage generateMasterKey()

```

86 public static byte[] generateMasterKey(byte[] password) throws NoSuchAlgorithmException {
87 3 return password.length > 0 ? MacDigestManager.genDigest("ledger", password) : MacDigestManager.EMPTY_LEDGER_KEY;
88 }
89

```

Figura 25 Mutation coverage generateMasterKey()

BinaryData

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
compare(BinaryData.Decoders.Schema)		0%		0%	26	26	63	63	1	1
encodeLong(long, byte[], int)		0%		0%	10	10	31	31	1	1
hashCode(BinaryData.HashData.Schema)		0%		0%	18	18	28	28	1	1
encodeDouble(double, byte[], int)		0%		n/a	1	1	12	12	1	1
encodeInt(int, byte[], int)		26%		12%	4	5	11	16	0	1
hashBytes(int, BinaryData.HashData, int, boolean)		0%		0%	4	4	11	11	1	1
encodeFloat(float, byte[], int)		0%		n/a	1	1	6	6	1	1
compare(byte[], int, int, byte[], int, int, Schema)		0%		n/a	1	1	6	6	1	1
hashCode(byte[], int, int, Schema)		0%		n/a	1	1	5	5	1	1
compare(byte[], int, byte[], int, Schema)		0%		n/a	1	1	1	1	1	1
skipLong(byte[], int)		0%		0%	2	2	2	2	1	1
encodeBoolean(boolean, byte[], int)		0%		0%	2	2	2	2	1	1
compareBytes(byte[], int, int, byte[], int, int)		100%		83%	1	4	0	8	0	1
static {...}		100%		n/a	0	1	0	2	0	1
Total	930 of 1.005	7%	97 of 103	5%	72	77	178	193	11	14

Figura 26 Statement e branch coverage suite minimale BinaryData

BinaryData

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
compare(BinaryData.Decoders.Schema)		0%		0%	26	26	63	63	1	1
encodeLong(long, byte[], int)		0%		0%	10	10	31	31	1	1
hashCode(BinaryData.HashData.Schema)		0%		0%	18	18	28	28	1	1
encodeDouble(double, byte[], int)		0%		n/a	1	1	12	12	1	1
hashBytes(int, BinaryData.HashData, int, boolean)		0%		0%	4	4	11	11	1	1
encodeFloat(float, byte[], int)		0%		n/a	1	1	6	6	1	1
compare(byte[], int, int, byte[], int, int, Schema)		0%		n/a	1	1	6	6	1	1
hashCode(byte[], int, int, Schema)		0%		n/a	1	1	5	5	1	1
compare(byte[], int, byte[], int, Schema)		0%		n/a	1	1	1	1	1	1
skipLong(byte[], int)		0%		0%	2	2	2	2	1	1
encodeBoolean(boolean, byte[], int)		0%		0%	2	2	2	2	1	1
encodeInt(int, byte[], int)		100%		100%	0	5	0	16	0	1
compareBytes(byte[], int, int, byte[], int, int)		100%		100%	0	4	0	8	0	1
static {...}		100%		n/a	0	1	0	2	0	1
Total	865 of 1.005	13%	89 of 103	13%	67	77	167	193	11	14

Figura 27 Statement e branch coverage suite estesa BinaryData

```

179.  /**
180.   * Lexicographically compare bytes. If equal, return zero. If greater-than,
181.   * return a positive value, if less than return a negative value.
182.   */
183.  public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
184.      int end1 = s1 + l1;
185.      int end2 = s2 + l2;
186.      for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
187.          int a = (b1[i] & 0xff);
188.          int b = (b2[j] & 0xff);
189.          if (a != b) {
190.              return a - b;
191.          }
192.      }
193.      return l1 - l2;
194.  }

```

Figura 28 Statement e branch coverage suite minimale compareBytes()

```

178.
179. /**
180.  * Lexicographically compare bytes. If equal, return zero. If greater-than,
181.  * return a positive value, if less than return a negative value.
182.  */
183. public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
184.     int end1 = s1 + l1;
185.     int end2 = s2 + l2;
186.     for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
187.         int a = (b1[i] & 0xff);
188.         int b = (b2[j] & 0xff);
189.         if (a != b) {
190.             return a - b;
191.         }
192.     }
193.     return l1 - l2;
194. }

```

Figura 29 Statement e branch coverage suite estesa compareBytes()

```

179 /**
180  * Lexicographically compare bytes. If equal, return zero. If greater-than,
181  * return a positive value, if less than return a negative value.
182  */
183 public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
184 1 int end1 = s1 + l1;
185 1 int end2 = s2 + l2;
186 6 for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
187 1 int a = (b1[i] & 0xff);
188 1 int b = (b2[j] & 0xff);
189 1 if (a != b) {
190 2 return a - b;
191     }
192 }
193 2 return l1 - l2;
194 }
195

```

Figura 30 Mutation coverage suite minimale compareBytes()

```

179 /**
180  * Lexicographically compare bytes. If equal, return zero. If greater-than,
181  * return a positive value, if less than return a negative value.
182  */
183 public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
184 1 int end1 = s1 + l1;
185 1 int end2 = s2 + l2;
186 6 for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
187 1 int a = (b1[i] & 0xff);
188 1 int b = (b2[j] & 0xff);
189 1 if (a != b) {
190 2 return a - b;
191     }
192 }
193 2 return l1 - l2;
194 }

```

Figura 31 Mutation coverage suite estesa compareBytes()

```

309.  /**
310.   * Encode an integer to the byte array at the given position. Will throw
311.   * IndexOutOfBoundsException if it overflows. Users should ensure that there are at least
312.   * 5 bytes left in the buffer before calling this method.
313.   *
314.   * @return The number of bytes written to the buffer, between 1 and 5.
315.   */
316.  public static int encodeInt(int n, byte[] buf, int pos) {
317.      // move sign to low-order bit, and flip others if negative
318.      n = (n << 1) ^ (n >> 31); // n*2 invertito se negativo
319.      int start = pos;
320.      ◆ if ((n & ~0x7F) != 0) { //1000.0000 n >= 128
321.          buf[pos++] = (byte) ((n | 0x80) & 0xFF); // n or 1000.0000 and 1111.1111
322.          n >>= 7; //
323.          ◆ if (n > 0x7F) {
324.              buf[pos++] = (byte) ((n | 0x80) & 0xFF);
325.              n >>= 7;
326.              ◆ if (n > 0x7F) {
327.                  buf[pos++] = (byte) ((n | 0x80) & 0xFF);
328.                  n >>= 7;
329.                  ◆ if (n > 0x7F) {
330.                      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
331.                      n >>= 7;
332.                  }
333.              }
334.          }
335.      }
336.      buf[pos++] = (byte) n;
337.      return pos - start;
338.  }

```

Figura 32 Statement e branch coverage suite minimale encodeInt()

```

309.  /**
310.   * Encode an integer to the byte array at the given position. Will throw
311.   * IndexOutOfBoundsException if it overflows. Users should ensure that there are at least
312.   * 5 bytes left in the buffer before calling this method.
313.   *
314.   * @return The number of bytes written to the buffer, between 1 and 5.
315.   */
316.  public static int encodeInt(int n, byte[] buf, int pos) {
317.      // move sign to low-order bit, and flip others if negative
318.      n = (n << 1) ^ (n >> 31); // n*2 invertito se negativo
319.      int start = pos;
320.      ◆ if ((n & ~0x7F) != 0) { //1000.0000 n >= 128
321.          buf[pos++] = (byte) ((n | 0x80) & 0xFF); // n or 1000.0000 and 1111.1111
322.          n >>= 7; //
323.          ◆ if (n > 0x7F) {
324.              buf[pos++] = (byte) ((n | 0x80) & 0xFF);
325.              n >>= 7;
326.              ◆ if (n > 0x7F) {
327.                  buf[pos++] = (byte) ((n | 0x80) & 0xFF);
328.                  n >>= 7;
329.                  ◆ if (n > 0x7F) {
330.                      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
331.                      n >>= 7;
332.                  }
333.              }
334.          }
335.      }
336.      buf[pos++] = (byte) n;
337.      return pos - start;
338.  }

```

Figura 33 Statement e branch coverage suite estesa encodeInt()


```

309  /**
310   * Encode an integer to the byte array at the given position. Will throw
311   * IndexOutOfBoundsException if it overflows. Users should ensure that there are at least
312   * 5 bytes left in the buffer before calling this method.
313   *
314   * @return The number of bytes written to the buffer, between 1 and 5.
315   */
316  public static int encodeInt(int n, byte[] buf, int pos) {
317      // move sign to low-order bit, and flip others if negative
318  3 n = (n << 1) ^ (n >> 31); // n*2 invertito se negativo
319      int start = pos;
320  2 if ((n & ~0x7F) != 0) { // 1000.0000 n >= 128
321  3     buf[pos++] = (byte) ((n | 0x80) & 0xFF); // n or 1000.0000 and 1111.1111
322  1     n >>= 7;
323  2     if (n > 0x7F) {
324  3         buf[pos++] = (byte) ((n | 0x80) & 0xFF);
325  1         n >>= 7;
326  2         if (n > 0x7F) {
327  3             buf[pos++] = (byte) ((n | 0x80) & 0xFF);
328  1             n >>= 7;
329  2             if (n > 0x7F) {
330  3                 buf[pos++] = (byte) ((n | 0x80) & 0xFF);
331  1                 n >>= 7;
332             }
333         }
334     }
335 }
336 1 buf[pos++] = (byte) n;
337 2 return pos - start;
338 }

```

Figura 34 Mutation coverage suite minimale encodeInt()

```

309  /**
310   * Encode an integer to the byte array at the given position. Will throw
311   * IndexOutOfBoundsException if it overflows. Users should ensure that there are at least
312   * 5 bytes left in the buffer before calling this method.
313   *
314   * @return The number of bytes written to the buffer, between 1 and 5.
315   */
316  public static int encodeInt(int n, byte[] buf, int pos) {
317      // move sign to low-order bit, and flip others if negative
318  3 n = (n << 1) ^ (n >> 31); // n*2 invertito se negativo
319      int start = pos;
320  2 if ((n & ~0x7F) != 0) { // 1000.0000 n >= 128
321  3     buf[pos++] = (byte) ((n | 0x80) & 0xFF); // n or 1000.0000 and 1111.1111
322  1     n >>= 7;
323  2     if (n > 0x7F) {
324  3         buf[pos++] = (byte) ((n | 0x80) & 0xFF);
325  1         n >>= 7;
326  2         if (n > 0x7F) {
327  3             buf[pos++] = (byte) ((n | 0x80) & 0xFF);
328  1             n >>= 7;
329  2             if (n > 0x7F) {
330  3                 buf[pos++] = (byte) ((n | 0x80) & 0xFF);
331  1                 n >>= 7;
332             }
333         }
334     }
335 }
336 1 buf[pos++] = (byte) n;
337 2 return pos - start;
338 }

```

Figura 35 Mutation coverage suite estesa encodeInt()

SpecificData

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
createSchema(Type, Map)	<div><div></div></div>	8%	<div><div></div></div>	10%	24	25	44	48	0	1
SpecificData(ClassLoader)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	8	8	1	1
getProtocol(Class)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	8	8	1	1
getClass(Schema)	<div><div></div></div>	61%	<div><div></div></div>	64%	7	19	4	23	0	1
getNewRecordSupplier(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	7	7	1	1
newInstance(Class, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	7	7	1	1
createEnum(String, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	6	6	1	1
getForClass(Class)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	9	9	1	1
getNestedClassName(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
compare(Object, Object, Schema, boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
getForSchema(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	4	4	8	8	1	1
createFixed(Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
newRecord(Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
getSchema(Type)	<div><div></div></div>	33%	<div><div></div></div>	25%	2	3	3	5	0	1
getWrapper(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	6	6	7	7	1	1
getSchemaName(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
createString(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
lambda\$getNewRecordSupplier\$2(Class, Constructor, Object[], Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	3	3	1	1
getEnumSchema(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	1	1	1	1
lambda\$getClass\$0(Schema, String)	<div><div></div></div>	35%	<div><div></div></div>	n/a	0	1	4	5	0	1
isEnum(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	1	1	1	1
getDecoder(ObjectInput)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
getEncoder(ObjectOutput)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumReader(Schema, Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
lambda\$getSchema\$1(Type)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumWriter(Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumReader(Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
getClassName(Schema)	<div><div></div></div>	87%	<div><div></div></div>	50%	3	4	1	6	0	1
setCustomCoders(boolean)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	2	2	1	1
isStringType(Class)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
useCustomCoders()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
static {...}	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	9	0	1
SpecificData()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	8	0	1
isStringable(Class)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
get()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
Total	759 of 1.195	36%	116 of 141	17%	93	113	151	201	26	35

Figura 36 Statement e branch coverage suite minimale SpecificData(getClass())

SpecificData

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
createSchema(Type, Map)	<div><div></div></div>	8%	<div><div></div></div>	10%	24	25	44	48	0	1
SpecificData(ClassLoader)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	8	8	1	1
getProtocol(Class)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	8	8	1	1
getNewRecordSupplier(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	7	7	1	1
newInstance(Class, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	7	7	1	1
createEnum(String, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	6	6	1	1
getForClass(Class)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	9	9	1	1
compare(Object, Object, Schema, boolean)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
getForSchema(Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	4	4	8	8	1	1
createFixed(Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
newRecord(Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	4	4	1	1
getSchema(Type)	<div><div></div></div>	33%	<div><div></div></div>	25%	2	3	3	5	0	1
getSchemaName(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
createString(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	5	5	1	1
lambda\$getNewRecordSupplier\$2(Class, Constructor, Object[], Object, Schema)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	3	3	1	1
getClass(Schema)	<div><div></div></div>	87%	<div><div></div></div>	96%	1	19	1	23	0	1
getWrapper(Schema)	<div><div></div></div>	40%	<div><div></div></div>	16%	5	6	5	7	0	1
getEnumSchema(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	2	2	1	1	1	1
isEnum(Object)	<div><div></div></div>	0%	<div><div></div></div>	0%	3	3	1	1	1	1
getDecoder(ObjectInput)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
getEncoder(ObjectOutput)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumReader(Schema, Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
lambda\$getSchema\$1(Type)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumWriter(Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
createDatumReader(Schema)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
setCustomCoders(boolean)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	2	2	1	1
isStringType(Class)	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
useCustomCoders()	<div><div></div></div>	0%	<div><div></div></div>	n/a	1	1	1	1	1	1
getClassName(Schema)	<div><div></div></div>	93%	<div><div></div></div>	66%	2	4	0	6	0	1
static {...}	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	9	0	1
SpecificData()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	8	0	1
getNestedClassName(Schema)	<div><div></div></div>	100%	<div><div></div></div>	75%	1	3	0	5	0	1
lambda\$getClass\$0(Schema, String)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	5	0	1
isStringable(Class)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
get()	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	1	0	1
Total	687 of 1.195	42%	103 of 141	26%	83	113	136	201	24	35

Figura 37 Statement e branch coverage suite estesa SpecificData(getClass())

```

237.  /** Return the class that implements a schema, or null if none exists. */
238.  public Class getClass(Schema schema) {
239.  ♦ switch (schema.getType()) {
240.      case FIXED:
241.      case RECORD:
242.      case ENUM:
243.          String name = schema.getFullName();
244.  ♦ if (name == null)
245.      return null;
246.      Class c = classCache.computeIfAbsent(name, n -> {
247.          try {
248.              return ClassUtils.forName(getClassLoader(), getClassNames(schema));
249.          } catch (ClassNotFoundException e) {
250.              try { // nested class?
251.                  return ClassUtils.forName(getClassLoader(), getNestedClassNames(schema));
252.              } catch (ClassNotFoundException ex) {
253.                  return NO_CLASS;
254.              }
255.          }
256.      });
257.  ♦ return c == NO_CLASS ? null : c;
258.      case ARRAY:
259.          return List.class;
260.      case MAP:
261.          return Map.class;
262.      case UNION:
263.          List<Schema> types = schema.getTypes(); // elide unions with null
264.  ♦ if ((types.size() == 2) && types.contains(NULL_SCHEMA))
265.  ♦ return getWrapper(types.get(types.get(0).equals(NULL_SCHEMA) ? 1 : 0));
266.          return Object.class;
267.      case STRING:
268.  ♦ if (STRING_TYPE.STRING.equals(schema.getProp(STRING_PROP)))
269.          return String.class;
270.          return CharSequence.class;
271.      case BYTES:
272.          return ByteBuffer.class;
273.      case INT:
274.          return Integer.TYPE;
275.      case LONG:
276.          return Long.TYPE;
277.      case FLOAT:
278.          return Float.TYPE;
279.      case DOUBLE:
280.          return Double.TYPE;
281.      case BOOLEAN:
282.          return Boolean.TYPE;
283.      case NULL:
284.          return Void.TYPE;
285.      default:
286.          throw new AvroRuntimeException("Unknown type: " + schema);
287.      }
288.  }

```

Figura 38 Statement e branch coverage suite minimale getClass()

```

236.
237.  /** Return the class that implements a schema, or null if none exists. */
238.  public Class getClass(Schema schema) {
239.  ◆ switch (schema.getType()) {
240.      case FIXED:
241.      case RECORD:
242.      case ENUM:
243.          String name = schema.getFullName();
244.  ◆ if (name == null)
245.          return null;
246.          Class c = classCache.computeIfAbsent(name, n -> {
247.              try {
248.                  return ClassUtils.forName(getClassLoader(), getClassName(schema));
249.              } catch (ClassNotFoundException e) {
250.                  try { // nested class?
251.                      return ClassUtils.forName(getClassLoader(), getNestedClassName(schema));
252.                  } catch (ClassNotFoundException ex) {
253.                      return NO_CLASS;
254.                  }
255.              }
256.          });
257.  ◆ return c == NO_CLASS ? null : c;
258.      case ARRAY:
259.          return List.class;
260.      case MAP:
261.          return Map.class;
262.      case UNION:
263.          List<Schema> types = schema.getTypes(); // elide unions with null
264.  ◆ if ((types.size() == 2) && types.contains(NULL_SCHEMA))
265.  ◆     return getWrapper(types.get(types.get(0).equals(NULL_SCHEMA) ? 1 : 0));
266.          return Object.class;
267.      case STRING:
268.  ◆ if (STRING_TYPE.STRING.equals(schema.getProp(STRING_PROP)))
269.          return String.class;
270.          return CharSequence.class;
271.      case BYTES:
272.          return ByteBuffer.class;
273.      case INT:
274.          return Integer.TYPE;
275.      case LONG:
276.          return Long.TYPE;
277.      case FLOAT:
278.          return Float.TYPE;
279.      case DOUBLE:
280.          return Double.TYPE;
281.      case BOOLEAN:
282.          return Boolean.TYPE;
283.      case NULL:
284.          return Void.TYPE;
285.      default:
286.          throw new AvroRuntimeException("Unknown type: " + schema);
287.      }
288.  }

```

Figura 39 Statement e branch coverage suite estesa getClass()

```

237  /** Return the class that implements a schema, or null if none exists. */
238  public Class getClass(Schema schema) {
239      switch (schema.getType()) {
240          case FIXED:
241          case RECORD:
242          case ENUM:
243              String name = schema.getFullName();
244              if (name == null)
245                  return null;
246              Class c = classCache.computeIfAbsent(name, n -> {
247                  try {
248                      return ClassUtils.forName(getClassLoader(), getClassName(schema));
249                  } catch (ClassNotFoundException e) {
250                      try { // nested class?
251                          return ClassUtils.forName(getClassLoader(), getNestedClassName(schema));
252                      } catch (ClassNotFoundException ex) {
253                          return NO_CLASS;
254                      }
255                  }
256              });
257              return c == NO_CLASS ? null : c;
258          case ARRAY:
259              return List.class;
260          case MAP:
261              return Map.class;
262          case UNION:
263              List<Schema> types = schema.getTypes(); // elide unions with null
264              if ((types.size() == 2) && types.contains(NULL_SCHEMA))
265                  return getWrapper(types.get(types.get(0).equals(NULL_SCHEMA) ? 1 : 0));
266              return Object.class;
267          case STRING:
268              if (STRING_TYPE_STRING.equals(schema.getProp(STRING_PROP)))
269                  return String.class;
270              return CharSequence.class;
271          case BYTES:
272              return ByteBuffer.class;
273          case INT:
274              return Integer.TYPE;
275          case LONG:
276              return Long.TYPE;
277          case FLOAT:
278              return Float.TYPE;
279          case DOUBLE:
280              return Double.TYPE;
281          case BOOLEAN:
282              return Boolean.TYPE;
283          case NULL:
284              return Void.TYPE;
285          default:
286              throw new AvroRuntimeException("Unknown type: " + schema);
287          }
288      }

```

Figura 40 Mutation coverage getClass()

GenericData

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString(Object, StringBuilder, IdentityHashMap)		0%		0%	24	24	75	75	1	1
induce(Object)		0%		0%	21	21	44	44	1	1
validate(Schema, Object)		0%		0%	28	28	38	38	1	1
compare(Object, Object, Schema, boolean)		26%		18%	20	23	25	36	0	1
instanceOf(Schema, Object)		0%		0%	20	20	22	22	1	1
hashCode(Object, Schema)		0%		0%	12	12	20	20	1	1
writeEscapedString(CharSequence, StringBuilder)		0%		0%	16	16	26	26	1	1
getDefaultValue(Schema, Field)		0%		0%	7	7	19	19	1	1
getSchemaName(Object)		34%		35%	14	15	17	27	0	1
addLogicalTypeConversion(Conversion)		0%		0%	2	2	8	8	1	1
resolveUnion(Schema, Object)		38%		25%	6	7	8	14	0	1
deepCopy(Schema, Object)		31%		50%	2	4	6	11	0	1
newArray(Object, int, Schema)		0%		0%	3	3	7	7	1	1
deepCopyRaw(Schema, Object)		90%		91%	2	19	2	43	0	1
getConversionByClass(Class, LogicalType)		0%		0%	2	2	4	4	1	1
toString(Object)		0%		n/a	1	1	3	3	1	1
newMap(Object, int)		0%		0%	2	2	4	4	1	1
getFastReaderBuilder()		0%		0%	2	2	3	3	1	1
getConversionFor(LogicalType)		0%		0%	2	2	3	3	1	1
isFastReaderEnabled()		0%		0%	3	3	1	1	1	1
createFixed(Object, Schema)		47%		25%	2	3	1	3	0	1
newRecord(Object, Schema)		47%		25%	2	3	3	5	0	1
hashCodeAdd(int, Object, Schema)		0%		n/a	1	1	1	1	1	1
createDatumReader(Schema, Schema)		0%		n/a	1	1	1	1	1	1
compare(Object, Object, Schema)		0%		n/a	1	1	1	1	1	1
getConversionByClass(Class)		62%		50%	1	2	1	4	0	1
createDatumWriter(Schema)		0%		n/a	1	1	1	1	1	1
setFastReaderEnabled(boolean)		0%		n/a	1	1	2	2	1	1
createDatumReader(Schema)		0%		n/a	1	1	1	1	1	1
getConversions()		0%		n/a	1	1	1	1	1	1
getRecordSchema(Object)		0%		n/a	1	1	1	1	1	1
getEnumSchema(Object)		0%		n/a	1	1	1	1	1	1
getFixedSchema(Object)		0%		n/a	1	1	1	1	1	1
getArrayAsCollection(Object)		0%		n/a	1	1	1	1	1	1
isLong(Object)		0%		n/a	1	1	1	1	1	1
isFloat(Object)		0%		n/a	1	1	1	1	1	1
isDouble(Object)		0%		n/a	1	1	1	1	1	1
isBoolean(Object)		0%		n/a	1	1	1	1	1	1
getNewRecordSupplier(Schema)		0%		n/a	1	1	1	1	1	1
GenericData(ClassLoader)		94%		50%	1	2	0	8	0	1
createString(Object)		90%		75%	1	3	1	5	0	1

Figura 41 Statement e branch coverage GenericData (deepCopyRaw())

```

1235. private Object deepCopyRaw(Schema schema, Object value) {
1236.     if (value == null) {
1237.         return null;
1238.     }
1239.
1240.     switch (schema.getType()) {
1241.     case ARRAY:
1242.         List<Object> arrayValue = (List) value;
1243.         List<Object> arrayCopy = new GenericData.Array<>(arrayValue.size(), schema);
1244.         for (Object obj : arrayValue) {
1245.             arrayCopy.add(deepCopy(schema.getElementType(), obj));
1246.         }
1247.         return arrayCopy;
1248.     case BOOLEAN:
1249.         return value; // immutable
1250.     case BYTES:
1251.         ByteBuffer byteBufferValue = (ByteBuffer) value;
1252.         int start = byteBufferValue.position();
1253.         int length = byteBufferValue.limit() - start;
1254.         byte[] bytesCopy = new byte[length];
1255.         byteBufferValue.get(bytesCopy, 0, length);
1256.         ((Buffer) byteBufferValue).position(start);
1257.         return ByteBuffer.wrap(bytesCopy, 0, length);
1258.     case DOUBLE:
1259.         return value; // immutable
1260.     case ENUM:
1261.         return createEnum(value.toString(), schema);
1262.     case FIXED:
1263.         return createFixed(null, ((GenericFixed) value).bytes(), schema);
1264.     case FLOAT:
1265.         return value; // immutable
1266.     case INT:
1267.         return value; // immutable
1268.     case LONG:
1269.         return value; // immutable
1270.     case MAP:
1271.         Map<Object, Object> mapValue = (Map) value;
1272.         Map<Object, Object> mapCopy = new HashMap<>(mapValue.size());
1273.         for (Map.Entry<Object, Object> entry : mapValue.entrySet()) {
1274.             mapCopy.put(deepCopy(schema.getStringType(), entry.getKey()), deepCopy(schema.getValueType(), entry.getValue()));
1275.         }
1276.         return mapCopy;
1277.     case NULL:
1278.         return null;
1279.     case RECORD:
1280.         Object oldState = getRecordState(value, schema);
1281.         Object newRecord = newRecord(null, schema);
1282.         Object newState = getRecordState(newRecord, schema);
1283.         for (Field f : schema.getFields()) {
1284.             int pos = f.pos();
1285.             String name = f.name();
1286.             Object newValue = deepCopy(f.schema(), getField(value, name, pos, oldState));
1287.             setField(newRecord, name, pos, newValue, newState);
1288.         }
1289.         return newRecord;
1290.     case STRING:
1291.         return createString(value);
1292.     case UNION:
1293.         return deepCopy(schema.getTypes().get(resolveUnion(schema, value)), value);
1294.     default:
1295.         throw new AvroRuntimeException("Deep copy failed for schema \"" + schema + "\" and value \"" + value + "\"");
1296.     }
}

```

Figura 42 Statement e branch coverage deepCopyRaw()

```

1235 private Object deepCopyRaw(Schema schema, Object value) {
1236 1 if (value == null) {
1237     return null;
1238 }
1239
1240 switch (schema.getType()) {
1241 case ARRAY:
1242     List<Object> arrayValue = (List) value;
1243     List<Object> arrayCopy = new GenericData.Array<>(arrayValue.size(), schema);
1244     for (Object obj : arrayValue) {
1245         arrayCopy.add(deepCopy(schema.getElementType(), obj));
1246     }
1247 1 return arrayCopy;
1248 case BOOLEAN:
1249 1 return value; // immutable
1250 case BYTES:
1251     ByteBuffer byteBufferValue = (ByteBuffer) value;
1252     int start = byteBufferValue.position();
1253 1 int length = byteBufferValue.limit() - start;
1254     byte[] bytesCopy = new byte[length];
1255     byteBufferValue.get(bytesCopy, 0, length);
1256     ((Buffer) byteBufferValue).position(start);
1257 1 return ByteBuffer.wrap(bytesCopy, 0, length);
1258 case DOUBLE:
1259 1 return value; // immutable
1260 case ENUM:
1261 1 return createEnum(value.toString(), schema);
1262 case FIXED:
1263 1 return createFixed(null, ((GenericFixed) value).bytes(), schema);
1264 case FLOAT:
1265 1 return value; // immutable
1266 case INT:
1267 1 return value; // immutable
1268 case LONG:
1269 1 return value; // immutable
1270 case MAP:
1271     Map<Object, Object> mapValue = (Map) value;
1272     Map<Object, Object> mapCopy = new HashMap<>(mapValue.size());
1273     for (Map.Entry<Object, Object> entry : mapValue.entrySet()) {
1274         mapCopy.put(deepCopy(schema.getType(), entry.getKey()), deepCopy(schema.getValueType(), entry.getValue()));
1275     }
1276 1 return mapCopy;
1277 case NULL:
1278     return null;
1279 case RECORD:
1280     Object oldState = getRecordState(value, schema);
1281     Object newRecord = newRecord(null, schema);
1282     Object newState = getRecordState(newRecord, schema);
1283     for (Field f : schema.getFields()) {
1284         int pos = f.pos();
1285         String name = f.name();
1286         Object newValue = deepCopy(f.schema(), getField(value, name, pos, oldState));
1287 1 setField(newRecord, name, pos, newValue, newState);
1288     }
1289 1 return newRecord;
1290 case STRING:
1291 1 return createString(value);
1292 case UNION:
1293 1 return deepCopy(schema.getTypes().get(resolveUnion(schema, value)), value);
1294 default:
1295     throw new AvroRuntimeException("Deep copy failed for schema \"" + schema + "\" and value \"" + value + "\"");
1296 }

```

Figura 43 Mutation coverage suite minimale deepCopyRaw()

```

1235 private Object deepCopyRaw(Schema schema, Object value) {
1236 1 if (value == null) {
1237     return null;
1238 }
1239
1240 switch (schema.getType()) {
1241 case ARRAY:
1242     List<Object> arrayValue = (List) value;
1243     List<Object> arrayCopy = new GenericData.Array<>(arrayValue.size(), schema);
1244     for (Object obj : arrayValue) {
1245         arrayCopy.add(deepCopy(schema.getElementType(), obj));
1246     }
1247 1 return arrayCopy;
1248 case BOOLEAN:
1249 1 return value; // immutable
1250 case BYTES:
1251     ByteBuffer byteBufferValue = (ByteBuffer) value;
1252     int start = byteBufferValue.position();
1253 1 int length = byteBufferValue.limit() - start;
1254     byte[] bytesCopy = new byte[length];
1255     byteBufferValue.get(bytesCopy, 0, length);
1256     ((Buffer) byteBufferValue).position(start);
1257 1 return ByteBuffer.wrap(bytesCopy, 0, length);
1258 case DOUBLE:
1259 1 return value; // immutable
1260 case ENUM:
1261 1 return createEnum(value.toString(), schema);
1262 case FIXED:
1263 1 return createFixed(null, ((GenericFixed) value).bytes(), schema);
1264 case FLOAT:
1265 1 return value; // immutable
1266 case INT:
1267 1 return value; // immutable
1268 case LONG:
1269 1 return value; // immutable
1270 case MAP:
1271     Map<Object, Object> mapValue = (Map) value;
1272     Map<Object, Object> mapCopy = new HashMap<>(mapValue.size());
1273     for (Map.Entry<Object, Object> entry : mapValue.entrySet()) {
1274         mapCopy.put(deepCopy(schema.getStringType(), entry.getKey()), deepCopy(schema.getValueType(), entry.getValue()));
1275     }
1276 1 return mapCopy;
1277 case NULL:
1278     return null;
1279 case RECORD:
1280     Object oldState = getRecordState(value, schema);
1281     Object newRecord = newRecord(null, schema);
1282     Object newState = getRecordState(newRecord, schema);
1283     for (Field f : schema.getFields()) {
1284         int pos = f.pos();
1285         String name = f.name();
1286         Object newValue = deepCopy(f.schema(), getField(value, name, pos, oldState));
1287 1 setField(newRecord, name, pos, newValue, newState);
1288     }
1289 1 return newRecord;
1290 case STRING:
1291 1 return createString(value);
1292 case UNION:
1293 1 return deepCopy(schema.getTypes().get(resolveUnion(schema, value)), value);
1294 default:
1295     throw new AvroRuntimeException("Deep copy failed for schema \"" + schema + "\" and value \"" + value + "\"");
1296 }

```

Figura 44 Mutation coverage suite estesa deepCopyRaw()