Getting Started

For profiling the eviction, a simple application has been created. The main purpose of the application is to add new entries into the cache periodically with some intervals. The application is a simple Main which starts the separate java.util.TimerTask. The task works periodically and writes data to the cache according to the given portion (the measurements were taken for 1000 puts).

Each entry is 1kB, i.e. 10 bytes for a key and the rest for the value - 1014 bytes for the value.

Please note, that the measurements are taken the following way:

the maxEntries variable of the cache is set to 50000. I'm profiling 50000 puts before eviction and as soon as it is reached, the snapshot is made using JProfiler Snapshot tools, then the profiler is reseted and again the next 50000 puts are profiled (already after eviction). So later the comparison is made based on the profiled data for equal number of puts but before and after eviction start to see the difference.

The duration of puts given here is measured on separate machine without influence of other processes on it.

Prerequisites

The measurements were done for the following cache configurations:

1. ***LIRS/LRU + NON-TX/TX + No Store***
2. ***LIRS/LRU + NON-TX/TX + FileCacheStore + Passivation enabled/disabled***
3. ***LIRS/LRU + NON-TX/TX + JDBC Cache Store + Passivation enabled / disabled***

For measurements only FileCacheStore and JDBCCacheStore has been chosen for comparing the data for 2 different cache stores. Other cachestores may be used for profiling on request.

The Cache configuration used for all measurements is:

```
 new ConfigurationBuilder().jmxStatistics().enable().clustering()
 .cacheMode(CacheMode.LOCAL)
.transaction().transactionMode(TransactionMode.NON_TRANSACTIONAL/TRANSACTIONAL)
.storeAsBinary().storeKeysAsBinary(true).storeValuesAsBinary(true)
.eviction().maxEntries(50000)
.strategy(EvictionStrategy.LRU/LIRS)
.expiration().wakeUpInterval(5000)
//the store configuration would be defined in each section.
```

Please note, that during evaluation the following terms will be used:

Hot spot **-** the methods from org.infinispan package which invocation count or inherent time is changed;
Inherent time **-** how much time has been spent in the hot spot. All calls into this method are summed up regardless of the particular call sequence;
Invocations **-** the difference of **invocation count** of the hot spot;

The tables given below are the results of the comparison of 2 snapshots for the given configuration – the first snapshot is the one made before eviction starts but already 50000 puts are made, the 2nd one is made after eviction started and 50000 puts are done. There is some additional comparison, for which the description is given for which snapshots it is done.

Please note, that the document is constructed in the following way:

- *different cache configurations are discussed, during which for each of them the single put duration is given before and after eviction start;*
- *the CPU comparision tables are given for each of them, to see the increased inherent time and invocation count for called methods.*
- *the JDBC Probe comparison is done specifically for JDBC Cache Store, to see the queries which were executed during execution;*
- *the VM telemetry comparison is done to see the memory usage during eviction on/off;*
- *and finally the summary is written based on the information given in the whole document;*

LIRS/LRU + NON-/TRANSACTIONAL + No Store configuration

For this case no cache store is configured. This means that, after the cache is filled with entries with number “maxEntries”, the existing entries would be removed from the cache according to the Eviction strategy and replaced with the new ones.

LIRS NON-TRANSACTIONAL

The performance of the data insertion into cache is not changed before and after the entries eviction start. The duration is **~42ms** and ***~39 ms for 1000 puts*** before & after eviction correspondingly. These numbers are average for 50000 entries in the cache before and after eviction.

From the method calls perspective, in case of LIRS, as soon as the eviction is started the following methods call count increased:

Hot spot	Inherent time	Invocations
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.tempRemoveFromStack	**+107 ms (+424 %)**	**+283623**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRS.removeFromSegment	*+46,691µs (+140%)*	*+87372*

org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.hot	*+20,453 µs (+89 %)*	*+61347*
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.moveToStackTop	*+27,244 µs (+61 %)*	*+37918*
org.infinispan.container.DefaultDataContainer.purgeExpired	**+165 ms (+66 %)**	**+2**
org.infinispan.CacheImpl.put	***+2,063 ms (+68 %)***	***±0***

As you can see, after eviction started, the duration of CacheImpl.put() method is increased due to increased invocations number above. The most frequent method that is called is **org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.tempRemoveFromStack .**

LIRS TRANSACTIONAL

In case when the same configuration is measured but with TRANSACTIONAL cache, the view is completely different. The average duration of 1000 puts before and after eviction are correspondingly **~57ms and ~47ms**.

Only the following 2 methods have increased invocations count:

Hot spot	Inherent time	Invocations
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.moveToStackTop	***+31,498 µs (+61 %)***	***+36461***
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	***+68,479 µs (+87 %)***	***+36386***
org.infinispan.container.DefaultDataContainer.purgeExpired	+351 ms (+75 %)	+1

The considerable increase of Inherent Time is for the following method:

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	***+2,510 ms (+113 %)***	***±0***
org.infinispan.container.DefaultDataContainer.purgeExpired	+351 ms (+75 %)	+1
org.infinispan.transaction.TransactionCoordinator.prepare	+153 ms (+8 %)	±0

As you can see, the invocation number for commit() method is not increased. Only the duration of commit method is increased by 2.5s.

If comparing the **LIRS - NON-TRANSACTIONAL** and **TRANSACTIONAL** results, when the eviction is on and 50000 entries were put after eviction is started, the picture is:

Hot spot	Inherent time	Invocations
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	**+279 ms (+281 %)**	**+650000**
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	**+223 ms (+242 %)**	**+100000**
org.infinispan.CacheImpl.put	-2,621 ms (-51 %)	±0

As you can see, in case of TRANSACTIONAL cache, there is a huge increase in invocation count for Interceptor's methods, BUT as you can see from the table above, the **CacheImpl.put() method duration is 50% lower**.

LRU NON-TRANSACTIONAL
In this case the average duration of 1000 puts before and after eviction correspondingly are: **~41ms and ~49ms**.
The measurement results are:

Hot spot	Inherent time	Invocations
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LRU.createNewEntry	**+10,749 µs (+48 %)**	**+36754**
org.infinispan.CacheImpl.put	**+845 ms (+30 %)**	**±0**

So the comparison above is done for 2 snapshots one before eviction starts, another after eviction start. So after eviction start, the call to CacheImpl.put() is increased only by 845ms.

LRU - TRANSACTIONAL
Here the duration of 1000 puts before eviction is **~57ms and after eviction ~49ms.**
In this case, if comparing the data before and after eviction, the method's invocations number is not increased at all.

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	**+701 ms (+35 %)**	**±0**
org.infinispan.container.DefaultDataContainer.purgeExpired	+182 ms (+47 %)	-1

As you can see, only the duration of commit method is a bit increased, but no invocations increase is detected.

LIRS - TRANSACTIONAL vs NON-TRANSACTIONAL

Comparison is done between TX and NON_TX evictions for 50000 entries – EvictionStrategy.LIRS. The comparison results are:

Hot spot	Inherent time	Invocations
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	**+275 ms (+278 %)**	**+650000**
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	**+206 ms (+223 %)**	**+100000**
org.infinispan.interceptors.InterceptorChain.invoke	**+37,678 µs (+187 %)**	**+100000**
org.infinispan.CacheImpl.put	**-1,216 ms (-33 %)**	**±0**

As you can see there is invocation increase in Interceptors in case of TRANSACTIONAL cache, ***but the CacheImpl.put() method invocation time is decreased by 33%.***

LIRS/LRU + FileCacheStore

In this case, as an addition to the given cache configuration a FileCacheStore is added.
The cache configuration is:

```
<# confBuilderAbove #>
.loaders().passivation(true/false)
.preload(false)
.addFileCacheStore()
.purgeOnStartup(false)
.fetchPersistentState(false)
```

a) LRU + NON-TRANSACTIONAL + FileCacheStore + Passivation On/Off

The measurements are taken for both - **passivation enabled/disabled**.

Passivation On

While passivation is ON, the duration of **1000 puts last ~ 57ms** (before eviction). As soon as the eviction of entries is started, the duration of a single put is increased and is **~38.5ms, which means that 1000 puts take ~38474ms.**

During eviction, the call to the following method is t**he longest lasting method with increased invocation count (non-infinispan class):**

java.nio.channels.FileChannel.force	*+1,983 s (+571 %)*	*+85750*

The calls in org.infinispan package are distributed this way:

Hot spot	*Inherent time*	*Invocations*
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+13,712 ms (+679 %)**	**+5993760**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+6,496 ms (+734 %)**	**+5993760**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+3,238 ms (+682 %)**	**+2028710**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+2,002 ms (+695 %)**	**+1982525**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+3,019 ms (+664 %)**	**+1982525**
org.infinispan.loaders.file.FileCacheStore.loadBucket	-4,560 ms (-84 %)	+47817
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	+1,411 ms (+650 %)	+46185
org.infinispan.marshall.AbstractMarshaller.trimBuffer	+1,709 ms (+1287 %)	+46185
org.infinispan.container.DefaultDataContainer.purgeExpired	+959 ms (+187 %)	+285

As soon as the eviction starts, the puts with this configuration become very slow. As you can see from the table above, many hot spots were detected as with increased invocations count as well as with increased Inherent time.

Passivation Off

When the passivation is disabled the single put lasts **~45ms (before eviction start)**. As soon as the eviction is started the average time of single puts **is decreased until ~40ms**. So the difference is not so much. Comparing the snapshots taken for this configuration before and after eviction, the picture is:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+6,708 ms (+41 %)**	**-62088**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+3,170 ms (+45 %)**	**-62088**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	+1,542 ms (+43 %)	-20696

org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+1,428 ms (+39 %)	-20696
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	+1,033 ms (+45 %)	-20696

As you can see, no call invocations count is increased, vice versa the number of invocations is decreased, but the inherent time is increased.
For passivation enabled and disabled the top 2 hot spots are the same, although for the first one there is increased Invocations count and for disabled one there is no invocation count increase – only Inherent time increase.

b) LRU + TRANSACTIONAL + FileCacheStore + Passivation On/Off

Passivation On

Here the measurements are taken when the eviction strategy is LRU, the cache is TRANSACTIONAL. As a store a FileCacheStore is used and the passivation is ON.

When the passivation is ON, the duration of **the single put to cache before eviction start is about ~35ms. The single put after the eviction starts last ~77ms**.

The Hot spot comparison report given below is ordered by Inherent Time increase.

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+32,738 ms (+37 %)**	-49
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+19,926 ms (+42 %)**	-156959
org.infinispan.transaction.TransactionCoordinator.commit	**+8,650 ms (+86 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+4,072 ms (+27 %)**	+2371852
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+3,235 ms (+32 %)**	-14055712
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+2,815 ms (+39 %)**	-13787940
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+1,797 ms (+26 %)**	2371852

The next report given below is ordered by Invocations count increase:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+4,072 ms (+27 %)**	**+2371852**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+1,797 ms (+26 %)**	**+2371852**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+884 ms (+25 %)	**+808024**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	+581 ms (+26 %)	**+781914**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	+950 ms (+28 %)	**+781914**

As you can see, there are many methods which invocation count is increased very abruptly.
You may see, that the top hot spots with increased invocations count as well as have increased Inherent time are the same as for options analyzed above.

Passivation Off

When the passivation is disabled, **the single put lasts ~44ms before and ~42ms after eviction start**.
The Hot spot comparision report ordered by Inherent time is:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+4,781 ms (+14 %)**	**-62088**
org.infinispan.transaction.TransactionCoordinator.commit	**+3,316 ms (+31 %)**	**±0**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+2,116 ms (+13 %)**	-62088
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+1,090 ms (+14 %)**	-20696
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+1,068 ms (+13 %)**	-20696

Many methods Inherent time is increased for a few seconds, but no Invocation count increase I detected.

c) LIRS + NON-TRANSACTIONAL + FileCacheStore + Passivation On/Off

Passivation On

In this case the EvictionStrategy is set to LIRS. The cache is NON_TRANSACTIONAL, as a store the FileCacheStore is used and the passivation is enabled.

For this configuration, **the single put lasts ~20ms before and ~39ms after eviction starts.**

The Hot spot comparison report ordered by Invocation count is:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+21,005 ms (+107 %)**	**+6327872**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+9,353 ms (+107 %)**	**+6327872**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+4,846 ms (+107 %)**	**+2142004**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+2,993 ms (+107 %)**	**+2092934**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+4,858 ms (+110 %)**	**+2092934**
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+12,752 ms (+44 %)**	**+49073**
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	**+2,070 ms (+108 %)**	**+49,070**

For comparison see below the same table but from point of Inherent time decrease.

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+136 s (+125 %)**	**+14**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+77,720 ms (+130 %)**	**-7772**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+21,005 ms (+107 %)**	**+6327872**
org.infinispan.CacheImpl.put	**+20,464 ms (+108 %)**	**±0**
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+15,638 ms (+115 %)**	**-8581744**
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+12,752 ms (+44 %)**	**+49073**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+11,694 ms (+126 %)**	**-10761194**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+9,353 ms (+107 %)**	**+6327872**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+4,858 ms (+110 %)**	**+2092934**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+4,846 ms (+107 %)**	**+2142004**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+2,993 ms (+107 %)**	**+2092934**

The slowest methods are marked as red.

Passivation Off

In this case the EvictionStrategy is set to LIRS. The cache is NON_TRANSACTIONAL, as a store the FileCacheStore is used and the passivation is disabled.

For this configuration, ***the single put lasts ~23ms before and ~22-23ms after eviction starts.***

The Hot spot comparison ordered by Inherent time would be:

Hot spot	Inherent time	Invocations
org.infinispan.CacheImpl.put	**+5,881 ms (+49 %)**	**±0**

As you can see, in case when the passivaiton is disabled, no invocations count is detected. Only the put() method duration is increased for ~5ms.

d) LIRS + TRANSACTIONAL + FileCacheStore + Passivation On/Off

Passivation On

The difference of this configuration compared with the one above is that the cache is TRANSACTIONAL.

The duration of single put is ~35ms before and ~19ms after eviction start.

Below you can find the Hot spot comparison report for this configuration ordered by Inherent time. As you can see in table the first 2 methods' inherent time is decreased harshly.

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+211 s (+1051 %)**	**+358**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+115 s (+1312 %)**	**+863271**

org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+42,185 ms (+10259 %)**	+13218965
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+37,672 ms (+2352 %)**	+49073
org.infinispan.transaction.TransactionCoordinator.commit	**+25,499 ms (+565 %)**	±0
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+23,249 ms (+1331 %)**	+55993478
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+18,831 ms (+10071 %)**	+13218965
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+17,932 ms (+1108 %)**	+49255683
org.infinispan.CacheImpl.put	**+10,339 ms (+249 %)**	±0

From the point of Invocations increase, the view is:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+23,249 ms (+1331 %)	**+55993478**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+17,932 ms (+1108 %)	**+49255683**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+42,185 ms (+10259 %)	**+13218965**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+18,831 ms (+10071 %)	**+13218965**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+9,741 ms (+9464 %)	**+4467341**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	+9,615 ms (+9556 %)	**+4375812**
org.infinispan.util.concurrent.locks.StripedLock.getLock	+2,161 ms (+874 %)	**+1567008**
org.infinispan.util.concurrent.locks.StripedLock.hash	+2,042 ms (+842 %)	**+1567008**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	+1,839 ms (+812 %)	**+1567008**
org.infinispan.io.ExposedByteArrayOutputStream.getRawBuffer	+1,976 ms (+1544 %)	**+954800**
org.infinispan.loaders.bucket.Bucket.setBucketId	+2,217 ms (+1275 %)	**+863271**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+115 s (+1312 %)	**+863271**
org.infinispan.marshall.VersionAwareMarshaller.startObjectInput	+1,635 ms (+1316 %)	**+863271**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	+3,067 ms (+1345 %)	**+863271**
org.infinispan.marshall.jboss.ExtendedRiverUnmarshaller.finish	+1,671 ms (+1323 %)	**+863271**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+3,798 ms (+1875 %)	**+863151**
org.infinispan.loaders.AbstractCacheStore.safeClose	+1,169 ms (+1242 %)	**+863150**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	+2,363 ms (+955 %)	**+783309**
org.infinispan.loaders.LockSupportCacheStore.unlock	+1,057 ms (+825 %)	**+783308**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+2,770 ms (+879 %)	**+783308**
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	+1,371 ms (+275 %)	**+736317**
org.infinispan.loaders.LockSupportCacheStore.lockForReading	+1,155 ms (+1094 %)	**+734235**
org.infinispan.loaders.file.FileCacheStore.loadBucket	+37,672 ms (+2352 %)	**+49073**
org.infinispan.container.EntryFactoryImpl.wrapEntryForPut	+519 ms (+611 %)	**+42846**
org.infinispan.container.DefaultDataContainer.purgeExpired	+3,036 ms (+1150 %)	**+778**
org.infinispan.loaders.file.FileCacheStore.purgeInternal	+211 s (+1051 %)	**+358**
org.infinispan.CacheImpl.put	+10,339 ms (+249 %)	**±0**

Passivation Off

The difference of this configuration compared with the one above is that the cache is that the passivation is disabled here.

The duration of single put is ~23ms before and ~23ms after eviction start.

Below you can find the Hot spot comparison report for this configuration ordered by Inherent time. As you can see in table the first 2 methods' inherent time is decreased considerably.

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	**+13,126 ms (+102 %)**	**±0**
org.infinispan.CacheImpl.put	+1,307 ms (+10 %)	±0

As you can see in this case, only the duration of commit() and put() are increased. The invocations count is not increased (even there is a decrease in many calls).

LIRS vs LRU Comparison For FileStore

This comparison was done to see the main methods invocations and durations difference between LIRS and LRU strategies.
The comparison is done for both Passivation enabled & disabled. In this comparison the cache is NON_TRANSACTIONAL.

Passivation Off

The comparison shows that while the eviction is not started, the methods invocation count is not increased, as well as there is very small difference in inherent time.

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+739 ms (+5 %)**	±0
org.infinispan.CacheImpl.put	**+348 ms (+3 %)**	±0
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+246 ms (+3 %)**	±0

As soon as the eviction starts, the difference of method calls increase very abruptly.

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	-1,097 ms (-11 %)	**+14583156**
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	-4,302 ms (-28 %)	**+13705345**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+61,353 ms (+13750 %)**	**+250791**
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+17,467 ms (+3600 %)**	±0

Passivation On

In case when the passivation is enabled, the picture is different. The difference is huge for both – before and after eviction.
Before eviction the comparison looks like:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+90,549 ms (+496 %)**	+401
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+51,569 ms (+643 %)**	+1006174
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+27,874 ms (+2191 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+19,471 ms (+8313 %)**	+7813784
org.infinispan.CacheImpl.put	**+12,549 ms (+198 %)**	±0
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+12,082 ms (+772 %)**	+74254008
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+8,602 ms (+8002 %)**	+7813784
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+7,714 ms (+500 %)**	+69079779
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+4,457 ms (+7350 %)**	+2636908
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	**+4,355 ms (+7716 %)**	+2588438

After eviction the comparison is:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+12,082 ms (+772 %)	**+74254008**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+7,714 ms (+500 %)	**+69079779**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+19,471 ms (+8313 %)	**+7813784**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+8,602 ms (+8002 %)	**+7813784**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+4,457 ms (+7350 %)	**+2636908**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	+4,355 ms (+7716 %)	**+2588438**
org.infinispan.util.concurrent.locks.StripedLock.getLock	+834 ms (+370 %)	**+1827551**
org.infinispan.util.concurrent.locks.StripedLock.hash	+800 ms (+363 %)	**+1827551**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	+722 ms (+354 %)	**+1827551**
org.infinispan.io.ExposedByteArrayOutputStream.getRawBuffer	+824 ms (+738 %)	**+1054644**
org.infinispan.io.UnsignedNumeric.readUnsignedInt	+445 ms (+578 %)	**+1006174**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.Integer)	+399 ms (+629 %)	**+1006174**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.String)	+959 ms (+620 %)	**+1006174**
org.infinispan.marshall.AbstractDelegatingMarshaller.finishObjectInput	+463 ms (+616 %)	**+1006174**

org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+51,569 ms (+643 %)	**+1006174**
org.infinispan.marshall.AbstractDelegatingMarshaller.startObjectInput	+450 ms (+603 %)	**+1006174**
org.infinispan.marshall.VersionAwareMarshaller.finishObjectInput	+404 ms (+622 %)	**+1006174**
org.infinispan.marshall.VersionAwareMarshaller.startObjectInput	+685 ms (+652 %)	**+1006174**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.finishObjectInput	+432 ms (+617 %)	**+1006174**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	+1,313 ms (+648 %)	**+1006174**
org.infinispan.marshall.jboss.ExtendedRiverUnmarshaller.finish	+677 ms (+609 %)	**+1006174**
org.infinispan.loaders.AbstractCacheStore.safeClose	+555 ms (+652 %)	**+1005937**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+1,560 ms (+864 %)	**+1005937**
org.infinispan.loaders.LockSupportCacheStore.lockForReading	+372 ms (+388 %)	**+913551**
org.infinispan.loaders.LockSupportCacheStore.unlock	+403 ms (+351 %)	**+913551**
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	+1,918 ms (+409 %)	**+913551**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	+782 ms (+361 %)	**+913551**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+1,086 ms (+393 %)	**+913551**
org.infinispan.CacheImpl.put	+12,549 ms (+198 %)	**±0**

TRANSACTIONAL vs NON_TRANSACTIONAL

LRU + FileCacheStore

In case when the **passivation is disabled**, the comparison of method calls for LRU + FileCacheStore is done comparing Transactional and Non-transactional caches.

Here the comparison is done for the snapshots done before eviction starts.
The comparison ordered by Invocations count is:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+9,381 ms (+68 %)	**+4264701**
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+12,301 ms (+69 %)	**+4264640**
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	+145 ms (+5 %)	**+51756**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+1,589 ms (+83 %)	**+51709**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+965 ms (+63 %)	**+51709**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+51,338 ms (+57 %)	**+51708**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	+1,228 ms (+72 %)	**+51708**
org.infinispan.loaders.file.FileCacheStore.purgeInternal	+106 s (+59 %)	**+19**
org.infinispan.CacheImpl.put	+756 ms (+6 %)	**±0**

The same comparison but ordered by Inherent Time, is:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+106 s (+59 %)**	+19
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+51,338 ms (+57 %)**	+51708
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+19,085 ms (+118 %)**	±0
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+18,661 ms (+95 %)**	±0
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+12,301 ms (+69 %)**	+4264640
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+9,381 ms (+68 %)**	+4264701
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+8,601 ms (+121 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+4,367 ms (+120 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+4,260 ms (+115 %)**	±0

If comparing the same but in case of **passivation enabled**, the picture is the following:
Ordered by Inherent time decrease:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+69,922 ms (+383 %)**	+313
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+39,164 ms (+488 %)**	+816034

org.infinispan.loaders.file.FileCacheStore.loadBucket	**+21,785 ms (+1712 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+14,962 ms (+6388 %)**	+6210598
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+8,671 ms (+554 %)**	+55272006
org.infinispan.CacheImpl.put	**+6,970 ms (+110 %)**	±0
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+6,730 ms (+6261 %)**	+6210598
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+5,616 ms (+364 %)**	+50670003
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+3,535 ms (+5829 %)**	+2,098,894
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	**+3,346 ms (+5928 %)**	+2055852
org.infinispan.interceptors.locking.ClusteringDependentLogic$AllNodesLogic.commitEntry	**+1,626 ms (+339 %)**	±0
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	**+1,553 ms (+825 %)**	+850000
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	**+1,523 ms (+325 %)**	+723527
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	**+1,350 ms (+748 %)**	+815910
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	**+1,107 ms (+546 %)**	+816034
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	**+1,071 ms (+727 %)**	+100000

The same comparison but with ordering of invocations count increase is:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+8,671 ms (+554 %)	**+55272006**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+5,616 ms (+364 %)	**+50670003**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+14,962 ms (+6388 %)	**+6210598**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+6,730 ms (+6261 %)	**+6210598**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+3,535 ms (+5829 %)	**+2098894**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	+3,346 ms (+5928 %)	**+2055852**
org.infinispan.util.concurrent.locks.StripedLock.getLock	+698 ms (+310 %)	**+1447505**
org.infinispan.util.concurrent.locks.StripedLock.hash	+684 ms (+310 %)	**+1447505**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	+593 ms (+291 %)	**+1447505**
org.infinispan.io.ExposedByteArrayOutputStream.getRawBuffer	+661 ms (+592 %)	**+859076**
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	+1,553 ms (+825 %)	**+850000**
org.infinispan.io.UnsignedNumeric.readUnsignedInt	+375 ms (+487 %)	**+816034**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.Integer)	+339 ms (+533 %)	**+816034**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.String)	+830 ms (+536 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.finishObjectInput	+385 ms (+512 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+39,164 ms (+488 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.startObjectInput	+381 ms (+511 %)	**+816034**
org.infinispan.marshall.VersionAwareMarshaller.finishObjectInput	+336 ms (+518 %)	**+816034**
org.infinispan.marshall.VersionAwareMarshaller.startObjectInput	+614 ms (+585 %)	**+816034**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.finishObjectInput	+363 ms (+519 %)	**+816034**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	+1,107 ms (+546 %)	**+816034**
org.infinispan.marshall.jboss.ExtendedRiverUnmarshaller.finish	+561 ms (+505 %)	**+816034**
org.infinispan.loaders.AbstractCacheStore.safeClose	+476 ms (+559 %)	**+815910**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+1,350 ms (+748 %)	**+815910**
org.infinispan.loaders.LockSupportCacheStore.lockForReading	+335 ms (+351 %)	**+723527**
org.infinispan.loaders.LockSupportCacheStore.unlock	+337 ms (+294 %)	**+723527**
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	+1,523 ms (+325 %)	**+723527**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	+645 ms (+298 %)	**+723527**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+853 ms (+309 %)	**+723527**
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	+1,071 ms (+727 %)	**+100000**
org.infinispan.container.EntryFactoryImpl.wrapEntryForPut	+390 ms (+513 %)	**+43491**
org.infinispan.container.DefaultDataContainer.purgeExpired	+585 ms (+250 %)	**+313**

org.infinispan.loaders.file.FileCacheStore.purgeInternal	+69,922 ms (+383 %)	**+313**
org.infinispan.CacheImpl.getInvocationContextWithImplicitTransaction	+357 ms (+446 %)	**±0**

When the **eviction is started**, the same comparison will look like:
Ordered by Inherent time:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+122 s (+27503 %)**	+351345
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+34,675 ms (+7147 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+17,159 ms (+75 %)**	±0
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+10,420 ms (+67 %)**	+28977568
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+9,389 ms (+98 %)**	+28977747
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+7,547 ms (+73 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+3,923 ms (+76 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+3,893 ms (+75 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+2,424 ms (+72 %)**	±0
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	**+1,513 ms (+71 %)**	±0
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	**+1,249 ms (+64 %)**	+351345

The same comparison ordered by Invocations count is:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+9,389 ms (+98 %)	**+28977747**
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+10,420 ms (+67 %)	**+28977568**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+1,249 ms (+64 %)	**+351345**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+122 s (+27503 %)	**+351345**

The same comparison is performed for **Passivation enabled setting**. In case when the eviction is not yet started, the numbers are – as usual ordered by Inherent time:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+69,922 ms (+383 %)**	+313
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+39,164 ms (+488 %)**	+816034
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+21,785 ms (+1712 %)**	±0
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+14,962 ms (+6388 %)**	+6210598
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+8,671 ms (+554 %)**	+55272006
org.infinispan.CacheImpl.put	**+6,970 ms (+110 %)**	±0
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+6,730 ms (+6261 %)**	+6210598
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+5,616 ms (+364 %)**	+50670003
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+3,535 ms (+5829 %)**	+2098894
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	**+3,346 ms (+5928 %)**	+2055852
org.infinispan.interceptors.locking.ClusteringDependentLogic$AllNodesLogic.commitEntry	**+1,626 ms (+339 %)**	±0
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	**+1,553 ms (+825 %)**	+850000
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	**+1,523 ms (+325 %)**	+723527
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	**+1,350 ms (+748 %)**	+815910
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	**+1,107 ms (+546 %)**	+816034
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	**+1,071 ms (+727 %)**	+100000

Ordered by Invocations count increase:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+8,671 ms (+554 %)**	**+55272006**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+5,616 ms (+364 %)**	**+50670003**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+14,962 ms (+6388 %)	**+6210598**

org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+6,730 ms (+6261 %)	**+6210598**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+3,535 ms (+5829 %)	**+2098894**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject	+3,346 ms (+5928 %)	**+2055852**
org.infinispan.util.concurrent.locks.StripedLock.getLock	+698 ms (+310 %)	**+1447505**
org.infinispan.util.concurrent.locks.StripedLock.hash	+684 ms (+310 %)	**+1447505**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	+593 ms (+291 %)	**+1447505**
org.infinispan.io.ExposedByteArrayOutputStream.getRawBuffer	+661 ms (+592 %)	**+859076**
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	+1,553 ms (+825 %)	**+850000**
org.infinispan.io.UnsignedNumeric.readUnsignedInt	+375 ms (+487 %)	**+816034**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.Integer)	+339 ms (+533 %)	**+816034**
org.infinispan.loaders.bucket.Bucket.setBucketId(java.lang.String)	+830 ms (+536 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.finishObjectInput	+385 ms (+512 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	+39,164 ms (+488 %)	**+816034**
org.infinispan.marshall.AbstractDelegatingMarshaller.startObjectInput	+381 ms (+511 %)	**+816034**
org.infinispan.marshall.VersionAwareMarshaller.finishObjectInput	+336 ms (+518 %)	**+816034**
org.infinispan.marshall.VersionAwareMarshaller.startObjectInput	+614 ms (+585 %)	**+816034**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.finishObjectInput	+363 ms (+519 %)	**+816034**
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	+1,107 ms (+546 %)	**+816034**
org.infinispan.marshall.jboss.ExtendedRiverUnmarshaller.finish	+561 ms (+505 %)	**+816034**
org.infinispan.loaders.AbstractCacheStore.safeClose	+476 ms (+559 %)	**+815910**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+1,350 ms (+748 %)	**+815910**
org.infinispan.loaders.LockSupportCacheStore.lockForReading	+335 ms (+351 %)	**+723527**
org.infinispan.loaders.LockSupportCacheStore.unlock	+337 ms (+294 %)	**+723527**
org.infinispan.loaders.file.FileCacheStore$NumericNamedFilesFilter.accept	+1,523 ms (+325 %)	**+723527**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	+645 ms (+298 %)	**+723527**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+853 ms (+309 %)	**+723527**
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	+1,071 ms (+727 %)	**+100000**
org.infinispan.CacheImpl.put	+6,970 ms (+110 %)	**±0**

In case when the eviction is started, the table ordered by Inherent time would be:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+20,474 ms (+2244 %)**	**±0**
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+5,156 ms (+62 %)**	+25688122
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+4,865 ms (+95 %)**	+24427865
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	**+843 ms (+65 %)**	+402393

The same data ordered by Invocations count, would be:

Hot spot	Inherent time	Invocations
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	+5,156 ms (+62 %)	**+25688122**
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	+4,865 ms (+95 %)	**+24427865**
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+688 ms (+4 %)	**+1461368**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+141 ms (+2 %)	**+1461368**
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	+135 ms (+3 %)	**+494968**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	+9,778 µs (+0 %)	**+483200**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	+213 ms (+5 %)	**+483200**
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	+843 ms (+65 %)	**+402393**
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	+4,710 µs (+0 %)	**+11768**
org.infinispan.loaders.file.FileCacheStore.loadBucket	+20,474 ms (+2244 %)	**±0**

LIRS + FileCacheStore

Here I'm comparing the snapshots done for LIRS + FileCacheStore as a store. The tables below are given for **Passivation disabled setting**. Below you can find the comparison tables for period when the eviction is not started yet. This table shows as the increase of Inherent Time as well as only one of the methods' Invocation count is increased.

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	**+112 s (+75 %)**	**+68**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+55,773 ms (+74 %)**	-18375
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+25,130 ms (+148 %)**	±0
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+21,507 ms (+110 %)**	±0
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+13,003 ms (+91 %)**	-6425973
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+11,315 ms (+154 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+9,056 ms (+79 %)**	-6425910
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+5,831 ms (+157 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+5,795 ms (+156 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+3,726 ms (+159 %)**	±0
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	**+2,238 ms (+145 %)**	±0

When the eviction is started, the data is almost the same (both the increased inherent time and the increased invocations count are placed in the same table and there is only one method which invocations count is increased):

Hot spot	Inherent time	Invocations
org.infinispan.loaders.file.FileCacheStore.purgeInternal	+116 s (+106 %)	**+26**
org.infinispan.marshall.AbstractDelegatingMarshaller.objectFromObjectStream	**+59,863 ms (+97 %)**	-17550
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	**+15,340 ms (+95 %)**	-4895979
org.infinispan.container.entries.AbstractInternalCacheEntry.getKey	**+13,532 ms (+120 %)**	-5627089
org.infinispan.loaders.file.FileCacheStore.loadBucket	**+12,841 ms (+72 %)**	±0
org.infinispan.container.entries.ImmortalCacheEntry.canExpire	**+11,045 ms (+130 %)**	-2334160
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	**+7,046 ms (+101 %)**	-4895979
org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+3,659 ms (+103 %)**	-1631993
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+3,532 ms (+98 %)**	-1631993
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+2,326 ms (+104 %)**	-1631993
org.infinispan.loaders.file.FileCacheStore$BufferedFileSync.flush	**+1,932 ms (+116 %)**	-17550
org.infinispan.notifications.AbstractListenerImpl$ListenerInvocation.invoke	**+1,796 ms (+63 %)**	±0
org.infinispan.marshall.jboss.AbstractJBossMarshaller.startObjectInput	**+1,620 ms (+125 %)**	-17550
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	**+1,421 ms (+92 %)**	±0
org.infinispan.CacheImpl.put	**-2,903 ms (-16 %)**	±0

The same comparison is also done for the case, when the **passivation is enabled**.

Below you can see the comparison done for LIRS when the passivation is enabled, but the eviction is not started yet. You can see, that there is no inherent time increase. Only 2 methods call invocation count is increased.

Hot spot	Inherent time	Invocations
org.infinispan.interceptors.base.CommandInterceptor.invokeNextInterceptor	-39,720 µs (-6 %)	**+850000**
org.infinispan.interceptors.InvocationContextInterceptor.handleDefault	-306 ms (-41 %)	**+100000**
org.infinispan.CacheImpl.put	-14,738 ms (-78 %)	**±0**

When the eviction takes place, the view is:

Hot spot	Inherent time	Invocations
org.infinispan.marshall.jboss.ExternalizerTable.getObjectWriter	+1,885 ms (+5 %)	**-902582**
org.infinispan.marshall.jboss.JBossMarshaller$ExternalizerTableProxy.getObjectWriter	+954 ms (+5 %)	**-902582**

org.infinispan.marshall.jboss.ExternalizerTable$ExternalizerAdapter.writeObject	**+478 ms (+5 %)**	**-304792**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, org.infinispan.container.entries.ImmortalCacheEntry)	**+446 ms (+5 %)**	**-298895**
org.infinispan.container.entries.ImmortalCacheEntry$Externalizer.writeObject(java.io.ObjectOutput, java.lang.Object)	**+358 ms (+6 %)**	**-298895**
org.infinispan.loaders.bucket.Bucket$Externalizer.writeObject	**+152 ms (+4 %)**	**-5897**
org.infinispan.CacheImpl.put	**-24,862 ms (-63 %)**	**±0**

As you can see in this case, no method Invocations count is increased. Evenmore, the Inherent time is decreased for CacheImpl.put() method for about 24 seconds.

LIRS/LRU + JDBCCacheStore

Here the configuration is done so that as a store the JdbcCacheStore is used. Specifically as the key here is a string, the JdbcStringBasedCacheStore is used.

The configuration is:

```
.loaders().passivation(true/false).preload(false).addLoader(JdbcStringBasedCacheStoreConfigurationBuilder.class)
        .fetchPersistentState(false)
        .purgeOnStartup(false)
        .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_STRING_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BLOB")
        .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
        .connectionPool()
        .connectionUrl("jdbc:mysql://localhost:3306/test")
        .username("root")
        .driverClass("com.mysql.jdbc.Driver").build();
```

a) LIRS + NON-TRANSACTIONAL + JdbcCacheStore + Passivation On/Off

Passivation On

For this configuration before the eviction starts, the duration of 1000 puts last ~426 ms. But as soon as the eviction starts, the single put duration is increased and for next 50000 puts it is ~21ms in average.

The comparison before and after eviction looks like:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	**+12,806 ms (+6964 %)**	+49075
org.infinispan.CacheImpl.put	**+11,531 ms (+72 %)**	±0
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Statement)	**+2,047 ms (+120 %)**	+98328
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Connection)	**+1,126 ms (+43 %)**	+49252
org.infinispan.container.DefaultDataContainer.purgeExpired	**+1,024 ms (+283 %)**	+180

Here you can see that there is a big increase in Inherent Time duration for the methods given above.
The same table from the point of Invocations would be:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.logAfter	+80,793 µs (+38 %)	**+98504**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.logBefore	+69,607 µs (+40 %)	**+98504**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Statement)	+2,047 ms (+120 %)	**+98328**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.getLockFromKey	+201 ms (+45 %)	**+98146**
org.infinispan.util.concurrent.locks.StripedLock.getLock	+157 ms (+55 %)	**+98146**
org.infinispan.util.concurrent.locks.StripedLock.hash	+134 ms (+43 %)	**+98146**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	+122 ms (+47 %)	**+98146**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRS.removeFromSegment	+199 ms (+155 %)	**+83921**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Connection)	+1,126 ms (+43 %)	**+49252**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.releaseConnection	+176 ms (+42 %)	**+49252**

org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	+12,806 ms (+6964 %)	**+49075**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.ResultSet)	+165 ms (+109 %)	**+49074**
org.infinispan.loaders.LockSupportCacheStore.unlock	+59,778 µs (+34 %)	**+49073**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.getStringMapping	+124 ms (+48 %)	**+49073**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.isSupportedType	+53,540 µs (+43 %)	**+49073**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	+122 ms (+45 %)	**+49073**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	+173 ms (+41 %)	**+49073**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.moveToStackTop	+58,922 µs (+37 %)	**+34993**
org.infinispan.CacheImpl.put	+11,531 ms (+72 %)	**±0**

Passivation Off

Here the same configuration is used. Only the passivation is disabled.
For this configuration, the duration of single put is ~21ms before and after eviction start.

The comparison of snapshots before and after eviction for this setting is:

Ordered by Inherent Time:

Hot spot	Inherent time	Invocations
org.infinispan.CacheImpl.put	**+3,008 ms (+14 %)**	**-14**
org.infinispan.container.DefaultDataContainer.purgeExpired	**+487 ms (+52 %)**	**-53**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRS.removeFromSegment	**+120 ms (+73 %)**	**+83902**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	+64,256 µs (+19 %)	+34832
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.moveToStackTop	+22,608 µs (+12 %)	+34979

Ordered by Invocation Count:

Hot spot	Inherent time	Invocations
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRS.removeFromSegment	**+120 ms (+73 %)**	**+83902**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.moveToStackTop	**+22,608 µs (+12 %)**	**+34979**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	**+64,256 µs (+19 %)**	**+34832**

As you can see in case when eviction starts, the invocation number of the methods showed above is increased

b) LIRS + TRANSACTIONAL + JdbcCacheStore + Passivation On/Off

The configuration here is the same as described above, only the cache is TRANSACTIONAL.

Passivation On

For this configuration, the duration of 1000 puts is ~458ms before eviction start. After eviction start, the single put duration increases and takes ~21ms.

The Hot spot comparison for snapshots done before and after eviction start, shows the following.

Ordered by Inherent time:

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	**+10,875 ms (+190 %)**	**±0**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	**+10,811 ms (+8866 %)**	**+49075**
org.infinispan.CacheImpl.put	**+2,851 ms (+42 %)**	**±0**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Statement)	**+2,045 ms (+179 %)**	+98503
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Connection)	**+1,901 ms (+106 %)**	+49427

Ordered by Invocation count:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.logAfter	**+140 ms (+113 %)**	**+98854**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Statement)	**+2,045 ms (+179 %)**	**+98503**
org.infinispan.util.concurrent.locks.StripedLock.getLock	**+211 ms (+118 %)**	**+98146**

org.infinispan.util.concurrent.locks.StripedLock.hash	**+229 ms (+120 %)**	**+98,146**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	**+199 ms (+119 %)**	**+98,146**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Connection)	**+1,901 ms (+106 %)**	**+49,427**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.releaseConnection	**+278 ms (+99 %)**	**+49,427**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	**+10,811 ms (+8866 %)**	**+49,075**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.getLockFromKey	**+234 ms (+120 %)**	**+49,073**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.getStringMapping	**+202 ms (+135 %)**	**+49,073**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	**+194 ms (+117 %)**	**+49,073**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	**+293 ms (+112 %)**	**+49,073**
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	**+229 ms (+135 %)**	**+34,846**

Passivation Off

For this configuration the single put before and after eviction start lasts ~21ms.
The comparison of snapshots before and after eviction is the following (ordered by Inherent Time):

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	**+4,946 ms (+35 %)**	-11
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	**+131 ms (+40 %)**	+34836

The same data ordered by Invocations increase would be:

Hot spot	Inherent time	Invocations
org.infinispan.util.concurrent.BoundedConcurrentHashMap$LIRSHashEntry.miss	+131 ms (+40 %)	**+34836**

c) LRU + TRANSACTIONAL + JdbcCacheStore + Passivation On/Off

Passivation On

In this case the configuration is almost the same as described in above casses, but the EvictionStrategy is LRU. The cache is TRANSACTIONAL, JDBC cache store is used, as well as the passivation is enabled.

The duration of 1000 puts before eviction starts lasts ~441ms, but as soon as eviction starts, the duration of single put increases till ~21ms, so 1000 puts are ~21000ms.

The Hot spot comparison ordered by Inherent time would be:

Hot spot	Inherent time	Invocations
org.infinispan.container.DefaultDataContainer.purgeExpired	**+765 ms (+124 %)**	+85
org.infinispan.util.concurrent.locks.StripedLock.getLock	**+233 ms (+74 %)**	+97,410
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	**+225 ms (+83 %)**	+97,410
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	**+222 ms (+44 %)**	+48,705
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.getStringMapping	**+217 ms (+79 %)**	+48,705

The Hot spot comparison ordered by Invocation count would be:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.JdbcUtil.safeClose	**-1,213 ms (-69 %)**	**+97798**
org.infinispan.util.concurrent.locks.StripedLock.getLock	**+233 ms (+74 %)**	**+97410**
org.infinispan.util.concurrent.locks.StripedLock.hash	**+193 ms (+58 %)**	**+97410**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	**+225 ms (+83 %)**	**+97410**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.releaseConnection	**+10,257 µs (+2 %)**	**+48,788**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.getLockFromKey	**+189 ms (+54 %)**	**+48,705**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.getStringMapping	**+217 ms (+79 %)**	**+48,705**

As you can see, with this configuration there is no considerable increase in Inherent time – 0.5s, although increase in method invocation count is in place.

Passivation Off

In this case the configuration is almost the same as described in above casses, but the EvictionStrategy is LRU. The cache is TRANSACTIONAL, JDBC cache store is used, as well as the passivation is disabled.

For this configuraiton, the duration of single put before and after eviction start lasts ~21ms.
The Hot spot comparison ordered by Inherent time would be:

Hot spot	Inherent time	Invocations
org.infinispan.transaction.TransactionCoordinator.commit	+2,223 ms (+18 %)	±0

No considerable time increase is in this case (actually as expected – as before and after eviction start the data is written both to the memory and store).

From point of methods invocation count increase, nothing is detected. See the comparison data below:

Hot spot	Inherent time	Invocations
org.infinispan.commands.AbstractFlagAffectedCommand.hasFlag	**+5,782 µs (+2 %)**	**+1**

Only one method call is increased with 1, so nothing harmful.

c) LRU + NON-TRANSACTIONAL + JdbcCacheStore + Passivation On/Off

Passivation On

In this case the configuration is almost the same as described in above casses, but the EvictionStrategy is LRU. The cache is NON-TRANSACTIONAL, JDBC cache store is used, as well as the passivation is enabled.

Before eviction start, the duration of 1000 puts lasts ~425ms, but after eviction start, the duration of single put increases and lasts ~21ms.
The Hot spot comparison ordered by Inherent time would be:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	**+12,319 ms (+7540 %)**	**+49,075**
org.infinispan.CacheImpl.put	**+10,801 ms (+114 %)**	**±0**

As you can see the method call on the top is the one which execution time is increased considerably after eviction starts.
The same comparison ordered by Invocations count would be:

Hot spot	Inherent time	Invocations
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.logAfter	**+121 ms (+78 %)**	**+98726**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.logBefore	**+126 ms (+110 %)**	**+98,726**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Statement)	**+4,234 ms (+406 %)**	**+98,438**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.getLockFromKey	**+346 ms (+105 %)**	**+98,146**
org.infinispan.util.concurrent.locks.StripedLock.getLock	**+200 ms (+90 %)**	**+98,146**
org.infinispan.util.concurrent.locks.StripedLock.hash	**+215 ms (+91 %)**	**+98,146**
org.infinispan.util.concurrent.locks.StripedLock.hashToIndex	**+160 ms (+78 %)**	**+98,146**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.Connection)	**+1,922 ms (+103 %)**	**+49363**
org.infinispan.loaders.jdbc.connectionfactory.PooledConnectionFactory.releaseConnection	**+276 ms (+87 %)**	**+49363**
org.infinispan.loaders.jdbc.stringbased.JdbcStringBasedCacheStore.storeLockSafe	**+12,319 ms (+7540 %)**	**+49,075**
org.infinispan.loaders.jdbc.JdbcUtil.safeClose(java.sql.ResultSet)	**+147 ms (+94 %)**	**+49,074**
org.infinispan.loaders.LockSupportCacheStore.unlock	**+114 ms (+87 %)**	**+49,073**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.getStringMapping	**+209 ms (+104 %)**	**+49,073**
org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper.isSupportedType	**+96,315 µs (+96 %)**	**+49,073**
org.infinispan.util.concurrent.locks.StripedLock.acquireLock	**+188 ms (+88 %)**	**+49,073**
org.infinispan.util.concurrent.locks.StripedLock.releaseLock	**+263 ms (+80 %)**	**+49,073**
org.infinispan.CacheImpl.put	**+10,801 ms (+114 %)**	**±0**

Passivation Off

In this case the configuration is almost the same as described in above casses, but the EvictionStrategy is LRU. The cache is TRANSACTIONAL, JDBC cache store is used, as well as the passivation is disabled.

The duration of single put here lasts ~21ms before and after eviction start.

For this configuration no methods' invocation increase is detected – only some decreases happened, as well as only one method's execution time is increased, although this change is really insignificant.

Hot spot	Inherent time	Invocations
org.infinispan.container.DefaultDataContainer.purgeExpired	+67,109 µs (+7 %)	-83

JDBC Probe Measurement

The JDBC Probe measurement was done for LIRS & LRU strategies, with passivation enabled/disabled. The results are:

Passivation On

Please note, that the queries are the same for both LRU & LIRS, as well as there is almost no difference in execution time of the queries.
Before eviction start, the view of queries is executed on DB is:

Hot spot	Inherent time	Average Time	Events
DELETE FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE ID_COLUMN = ?	15,772 ms (92 %)	2,630 µs	5,995
92.4% - 15,772 ms - 5,995 hot spot inv. without CPU recording			
SELECT ID_COLUMN, DATA_COLUMN FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE ID_COLUMN = ?	1,225 ms (7 %)	204 µs	5,995
7.2% - 1,225 ms - 5,995 hot spot inv. without CPU recording			
SHOW COLLATION	27,829 µs (0 %)	2,319 µs	12
0.2% - 27,829 µs - 12 hot spot inv. without CPU recording			
/* mysql-connector-java-5.1.20 (Revision: tonci.grgin@oracle.com-20111003110438-qfydx066wsbydkbw) */SHOW VARIABLES WHERE Variable_name ='language' OR Variable_name = 'net_write_timeout' OR Variable_name = 'interactive_timeout' OR Variable_name = 'wait_timeout' OR Variable_name = 'character_set_client' OR Variable_name = 'character_set_connection' OR Variable_name = 'character_set' OR Variable_name = 'character_set_server' OR Variable_name = 'tx_isolation' OR Variable_name = 'transaction_isolation' OR Variable_name = 'character_set_results' OR Variable_name = 'timezone' OR Variable_name = 'time_zone' OR Variable_name = 'system_time_zone' OR Variable_name = 'lower_case_table_names' OR Variable_name = 'max_allowed_packet' OR Variable_name = 'net_buffer_length' OR Variable_name = 'sql_mode' OR Variable_name = 'query_cache_type' OR Variable_name = 'query_cache_size' OR Variable_name = 'init_connect'	19,821 µs (0 %)	1,651 µs	12
0.1% - 19,821 µs - 12 hot spot inv. without CPU recording			
DELETE FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE TIMESTAMP_COLUMN< ? AND TIMESTAMP_COLUMN> 0	6,583 µs (0 %)	506 µs	13
0.0% - 6,583 µs - 13 hot spot inv. without CPU recording			
SELECT @@session.tx_isolation	5,456 µs (0 %)	363 µs	15
0.0% - 5,456 µs - 15 hot spot inv. without CPU recording			
/* mysql-connector-java-5.1.20 (Revision: tonci.grgin@oracle.com-20111003110438-qfydx066wsbydkbw) */SELECT @@session.auto_increment_increment	4,050 µs (0 %)	337 µs	12
0.0% - 4,050 µs - 12 hot spot inv. without CPU recording			
SHOW FULL TABLES FROM `test` LIKE 'ISPN_STRING_TABLE_evictionTestingCache'	580 µs (0 %)	580 µs	1
0.0% - 580 µs - 1 hot spot inv. without CPU recording			

And after eviction started, the insert query is also added to the execution list:

Hot spot	Inherent time	Average Time	Events
INSERT INTO `ISPN_STRING_TABLE_evictionTestingCache` (DATA_COLUMN, TIMESTAMP_COLUMN, ID_COLUMN) VALUES(?,?,?)	180 s (89 %)	30,892 µs	5,853

Passivation Off

For this configuration, also the JDBC Probe measurement was done. For passivation disabled, the data is written to DB with application start. So the measurements before and after eviction is almost the same, although the duration of all queries is increasing with entries number increase in DB. The longest taking query is the insert into the database.

Hot spot	Inherent time	Average Time	Events
INSERT INTO `ISPN_STRING_TABLE_evictionTestingCache` (DATA_COLUMN, TIMESTAMP_COLUMN, ID_COLUMN) VALUES(?,?,?)	156 s (98 %)	29,399 µs	5,315
98.2% - 156 s - 5,315 hot spot inv. without CPU recording			
SELECT ID_COLUMN, DATA_COLUMN FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE ID_COLUMN = ?	1,477 ms (0 %)	277 µs	5,316
0.9% - 1,477 ms - 5,316 hot spot inv. without CPU recording			
SELECT ID_COLUMN FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE ID_COLUMN = ?	1,000 ms (0 %)	188 µs	5,321
0.6% - 1,000 ms - 5,321 hot spot inv. without CPU recording			
DELETE FROM `ISPN_STRING_TABLE_evictionTestingCache` WHERE	243 ms (0 %)	7,176 µs	34

```
TIMESTAMP_COLUMN< ? AND TIMESTAMP_COLUMN> 0
■ 0.2% - 243 ms - 34 hot spot inv. without CPU recording
UPDATE `ISPN_STRING_TABLE_evictionTestingCache` SET DATA_COLUMN = ? , ■ 128 ms (0 %)     25,694   5
TIMESTAMP_COLUMN=? WHERE ID_COLUMN = ?                                                       µs
■ 0.1% - 128 ms - 5 hot spot inv. without CPU recording
```

Telemetry Comparison

I've also performed the VM telemetry comparison (especially for JDBC Cache Store configuration), which displays high-level data on thread activity and memory heap and garbage collection in the VM.

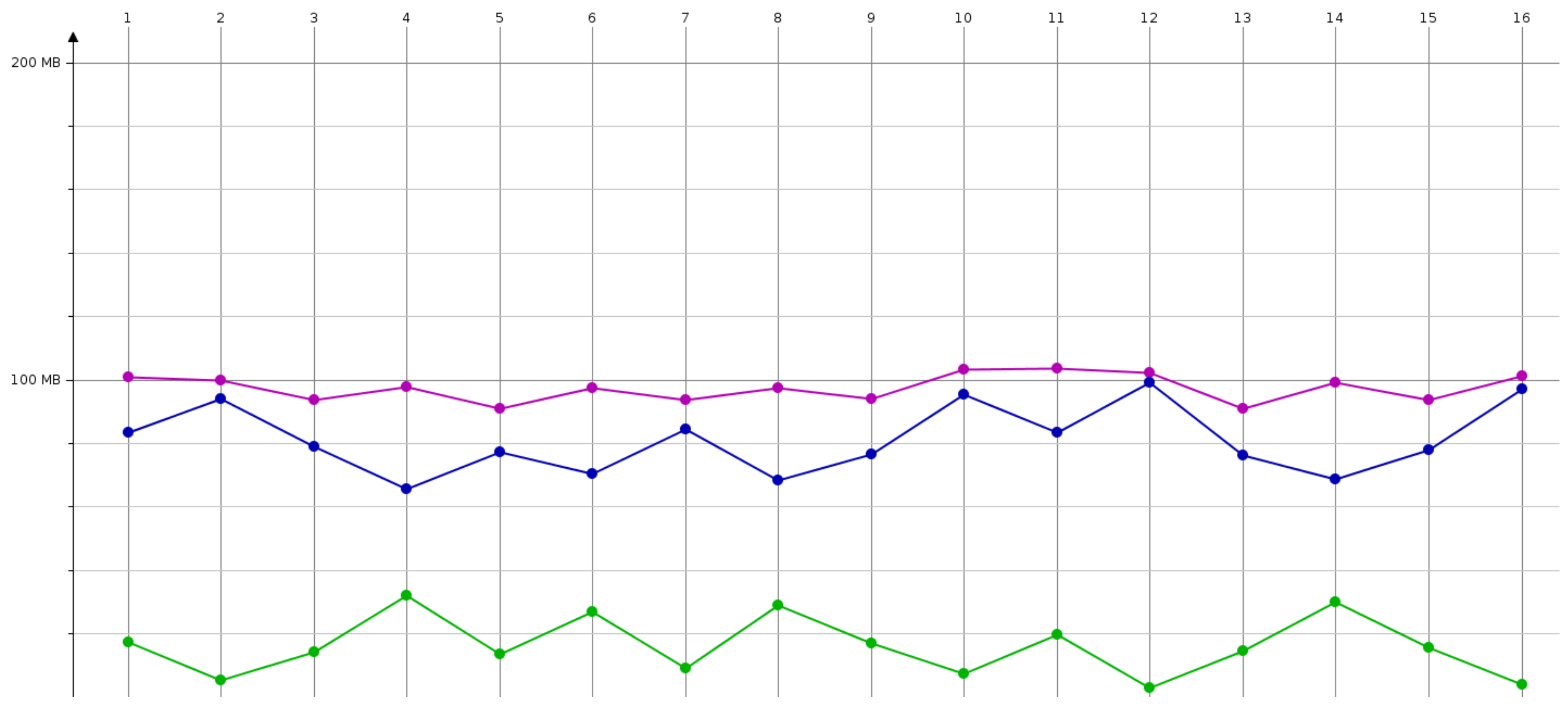
The comparison of Classes for snapshots made, shows that before eviction start the number of classes is ~700 in average for all type of configurations. After eviction start, the number of classes is increased for all configurations. This number is ~1900 for all configurations, except for LRU TRANSACTIONAL with passivation enabled for which this number is ~1000 classes, and for LRU NON-TX with passivation false, where the number is ~1200 classes. ***Please note that all numbers are given for the classes located on the Heap. For NON-HEAP memory, the number of classes for all configurations is almost the same.***

The Thread Comparison doesn't show any difference for neither of the configurations. So the number of threads is stable for all of them.

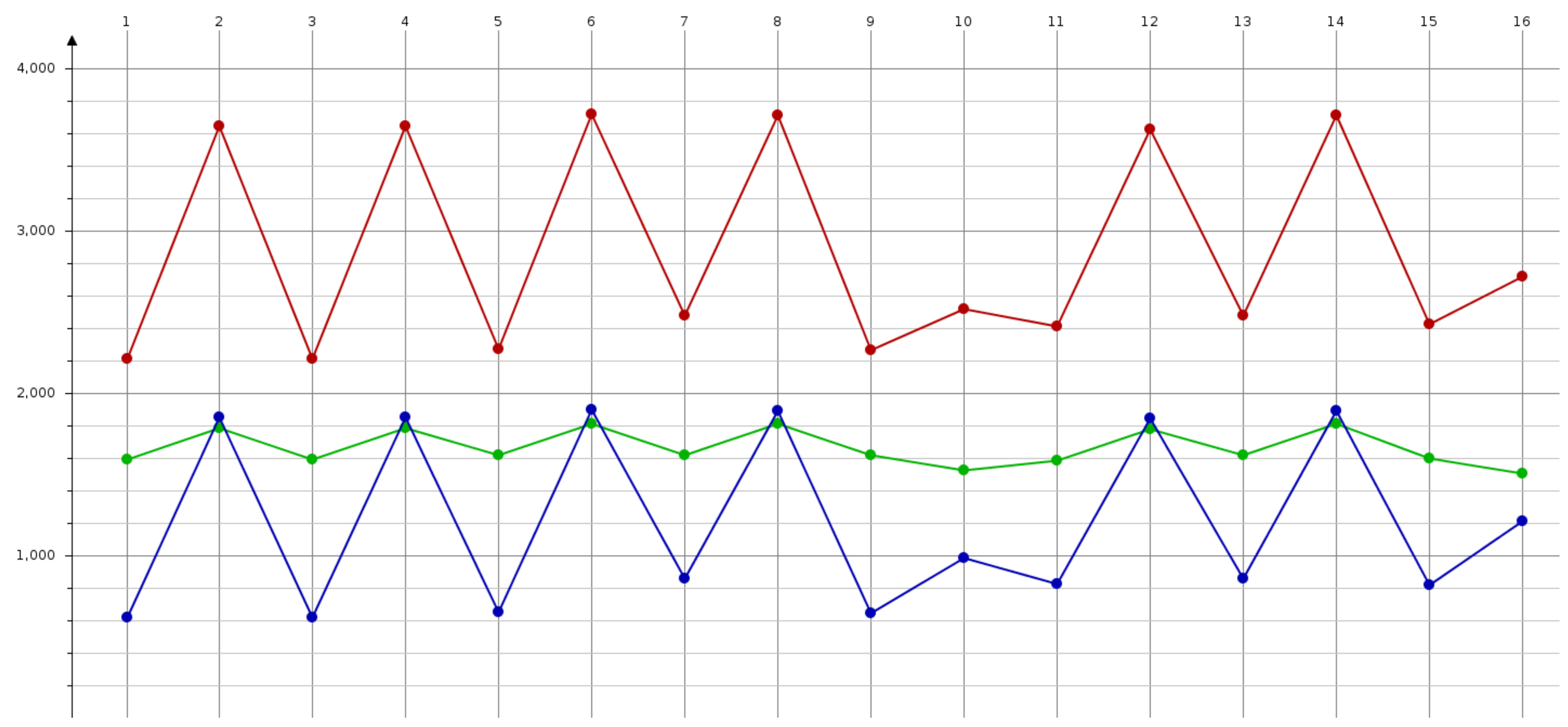
The memory comparison shows that in case when the eviction starts, there is more memory used rather than before eviction start, although this doesn't refer to the case when the passivation is disabled. In this case, the memory used is less for after eviction start than before.

Below you can see the memory comparison for the following configurations (all configurations are related to JDBC Cache store):

1. LIRS NON-TX, noeviction, passivation true
2. LIRS NON-TX, eviction passivation true
3. LIRS NON-TX, noeviction, passivation false
4. LIRS NON-TX, eviction, passivation false
5. LIRS TX, noeviction passivation false
6. LIRS TX, eviction, passivation false
7. LIRS TX, noeviction, passivation true
8. LIRS TX, eviction, passivation true
9. LRU TX, noeviction, passivation true
10. LRU TX, eviction, passivation true
11. LRU NON-TX, noeviction, passivation true
12. LRU NON-TX, eviction, passivation true
13. LRU TX, noeviction, passivation false
14. LRU TX, eviction, passivation false
15. LRU NON-TX, noeviction, passivation false
16. LRU NON-TX, eviction, passivation false



The class comparison graph would be (with the same coniguration order as given above):



Summary

As a summary, I can say that:

1. *The performance of the application, in case when the FileCacheStore is activated, is very slow.*
2. *In case when the passivation is disabled, the performance before and after eviction start, is the same. If we will compare the duration of single put before eviction start for passivation enabled and disabled, then we can see that for passivation disabled setting is much more non-perform-ant.* ***The duration of single put before eviction start for passivation disabled setting is direct comparative to the single put duration after eviction start in case of passivation enabled setting.*** *But, in this mode the JDBC Cache store is much more perform-ant than the File Cache Store.*
3. *In most cases, as soon as the eviction starts (especially in case of JDBCCacheStore):*
   1. *the performance of the puts is decreasing considerably. E.g. if the duration of 1000 puts is ~458ms, i.e. single put would be 0.458ms before eviction start, then after eviction takes place, the duration of single put is increased to 21ms;*
   2. *the number of classes in VM is increased considerably;*
   3. *the used HEAP memory is increased considerably as well;*

So as a final statement, I can say that when the eviction starts, the performance of the application is decreased. Moreover, if the eviction is enabled, but no entries are evicted yet, the performance of single put is worse rather than when the eviction is disabled.