# **HBASE-28292**

# Make Delay prefetch property to be dynamically configured

Problem:

Rolling restart triggers region movement on the cluster while the RegionServers are getting restarted. During this time, the temporary RegionServers will start prefetching these files which are only hosted until the source RegionServer is restarted. Hence, in this timing window, fetches are executed on temporary region servers which takes a few minutes.

Currently, this behavior is governed by hbase.hfile.prefetch.delay

Also, file-based block cache gives performance improvement on HBase clusters using object storage.

Proposed Solution

The number of prefetches on the cluster caused by these temporary region movements during Rolling Upgrade (rolling restart) can be minimized by introducing delay before the prefetch starts.

To achieve this, making the hbase.hfile.prefetch.delay dynamically configurable. The interval will let the rolling restart complete. Resultantly, prefetches on temporary region server can be minimized.

Proposed Change Set

<https://github.com/apache/hbase/pull/5605>

Review Discussion

During review, a case was discussed where

* 1. Prefetch delay is set to 60 mins initially then it’s changed to 5 seconds
  2. Now scheduled threads will wait for delay of 60 mins (starting from time when they were scheduled)
  3. As a result, despite the delay is changed, prefetched threads will wait
  4. <https://github.com/apache/hbase/pull/5605#discussion_r1444797276>

1. Is behavior mentioned in (1) acceptable?

2. If not, below is a proposal to address it.

* In the PrefetchExecutor class, ‘prefetchExecutorPool’ is initialized with 1 demon thread in a static block.
* When hfile block read request arrives, prefetch request is made in **HFilePreadReader**

| PrefetchExecutor.request(path, new Runnable() {  @Override  public void run() {  long offset = 0;  long end = 0;  HFile.Reader prefetchStreamReader = null;  ……… |
| --- |

* In the ‘run’ method of Runnable interface required context is provided to each thread. Hence, **HFilePreadReader** drives the prefetch and also calls cancel.
* The prefetch threads prefetching hfile blocks are managed using the ‘**prefetchFutures’** map**.**
* Proposal is to
  + add new map to store Runnable instances**.**
  + Populate it in **request**().

| **private static final Map<Path, Runnable> prefetchRunnable = new ConcurrentSkipListMap<>();** |
| --- |

* Now, when dynamic configuration is changed using hbase shell, **onConfigurationChange** method is called. This method will
  + call cancel on every prefetch thread
  + Update the new configuration value for hbase.hfile.prefetch.delay
  + Call **request()** on every element in the **prefetchRunnable**

| **@Override**  **public void onConfigurationChange(Configuration conf) {**  **PrefetchExecutor.loadConfiguration(conf);**  **}**  **public static void loadConfiguration(Configuration conf) {**  **prefetchDelayMillis = conf.getInt(PREFETCH\_DELAY, 1000);**  **prefetchFutures.forEach((k, v) -> {**  **cancel(k);**  **request(k,prefetchRunnable.get(k));**  **LOG.debug("Reset called on Prefetch of file {}", k);**  **});**  **}** |
| --- |

* Assumption here is, this way, every thread will have ‘context’ provided **in this code path**

| **HFilePreadReader**  **if (cacheConf.shouldPrefetchOnOpen() && cacheIfCompactionsOff() && shouldCache.booleanValue()) {**  **PrefetchExecutor.request(path, new Runnable() {**  **@Override**  **public void run() {**  **long offset = 0;**  **long end = 0;**  **HFile.Reader prefetchStreamReader = null;**  **try {**  **ReaderContext streamReaderContext = ReaderContextBuilder.newBuilder(context)**  **.withReaderType(ReaderContext.ReaderType.STREAM)**  **.withInputStreamWrapper(new FSDataInputStreamWrapper(context.getFileSystem(),**  **context.getInputStreamWrapper().getReaderPath())).build();**  **………………………….** |
| --- |

Please share your views on this one.