HBASE-10382

Problem statement

In the current design, every scanner creates a StoreScanner on the Stores associated with the table. During the course of a scan, if any new HFile is added to the store due to flush, compaction and bulk load, it is immediately added to the scanner's heap and the current scan tries to use the new files also in its scan. In order to support this, we have all the scanners created against this store registered with the HStore. Whenever any new file is added to the store, all the current ongoing scanners are notified about this and using a lock we reset the scanners' heap so that all the current ongoing scans can make use of the newly added files. As we can see that there is a need for a lock to be acquired to do this updation, every read operation like next(), seek() and reseek() tries to acquire the lock before proceeding with the operation.

This leads to a lot of lock acquiring and releasing step thus leading to a performance impact.

Solution

One solution to solve this – (illustrated by Lar's patch) where we try to move the lock from the StoreScanner to the RegionScanner level and pass down this RegionScanner level lock to these StoreScanners such that we don't need to acquire and release the lock every time. However still the problem suggested above still exists because when there is an updation to the files in the store happens we need to synchronize on the RegionScanner lock and do the necessary updations before proceeding with the scans. Also passing down this region lock to the StoreScanners require a change in the way the CPs are allowed to create their own Storescanners since the lock has to be passed to the CPs to make use of the region level lock.

The other solution is to have a **versioned Storefile approach**. What we mean by versioned approach is that the current ongoing scans will use the existing files for its scan to complete. Take the case of a compaction, suppose a scan is going through the 3 files that were involved in the compaction, the scan would still use those 3 files for the scan to be completed though in the back ground the 3 files have been compacted to a new file. Any new scanner that starts after the compaction is completed will only make use of the new file created out of compaction.

Doing this will avoid any locks in the StoreScanner and the need for any notification to all the registered scanners. This also ensures that the current ongoing scanners blocks that are in the block cache can effectively be used till the scanners are completed and there will be no need for any eviction of the blocks of the compacted files and reading the blocks of the newly compacted files from hdfs for the ongoing scan.

Implementation

For implementing this versioned approach, we introduce a ref counting mechanism on all the Storefile readers. Whenever a scanner is created on a store file we increment the ref count by 1. When a scan completes using the current file we decrement the ref count. Along with the ref count, we also have CompactionStatus (COMPACTED and NON_COMPACTED) added to these store file readers. Any file by default is in NON_COMPACTED state. Once a compaction completes we update the state to COMPACTED.

When a scanner tries to create a scanner on these storefiles, we check for the status of the file whether they are in COMPACTED or NON_COMPACTED state. If the state is NON_COMPACTED, we allow the scanner to be created on this file and also increment the ref count.

Any file which is already in the COMPACTED state is not allowed to be part of the scan because when the state is changed to COMPACTED it means that already the current file has been compacted and its content are copied to a new file which is in the NON_COMPACTED state and that scan will be using that new file instead of the files in the NON_COMPACTED state.

StoreFile Management

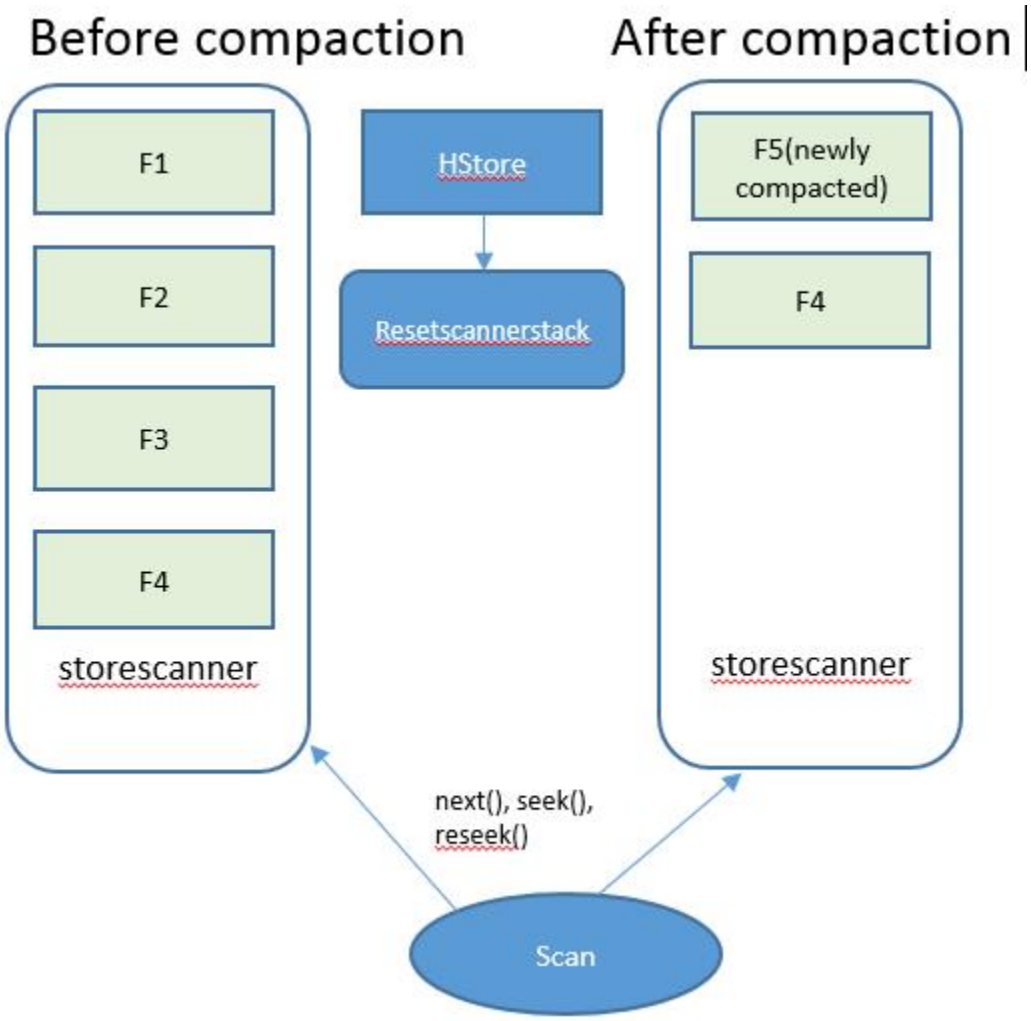
In the current design whenever a file is compacted, the compacted files are removed from the storefile list maintained by the Storefile manager. Thus at any point of time asking for a list of store files in the current store will only give the list of files that are not compacted. The compacted files are moved to archive directory.

In order to implement the versioned structure, we should ensure that the compacted files are not immediately removed from the list of storefiles maintained. On completion of a compaction mark the storefile as COMPACTED and do not archive the compacted store files. Ensure that a background thread runs periodically runs that scans the list of store files and checks for the state as COMPACTED and that there are no active readers on that file (ref count on those files should be 0), select all such files and remove them from the list of storefiles and move those files to the archive directory.

Impact on Split

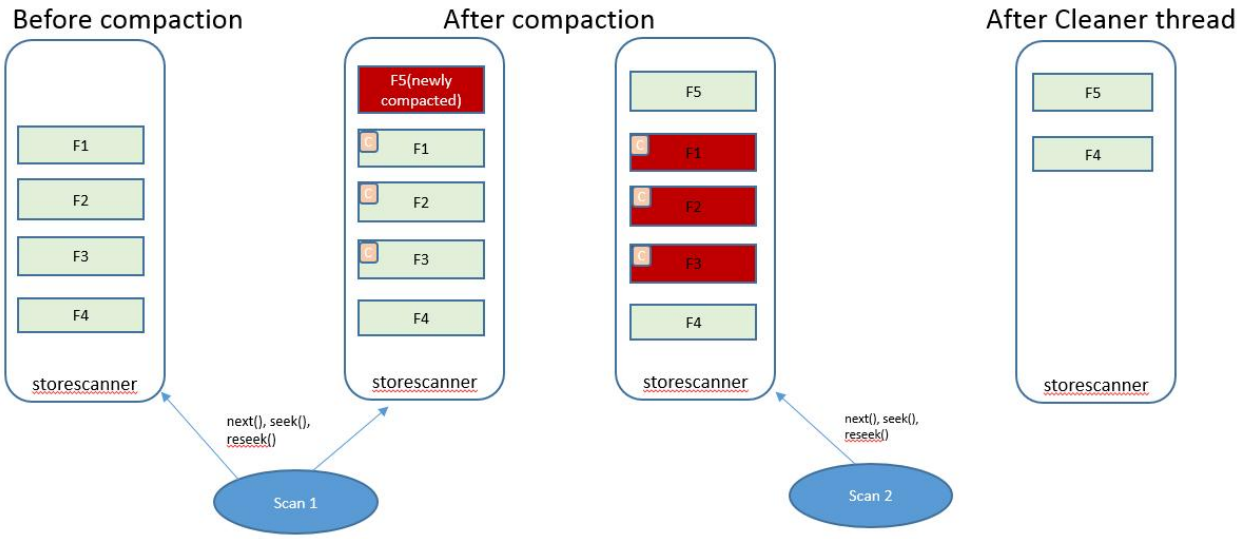
After split a compaction is performed to remove all the reference files. Any further splits will happen only if there are no reference files. Now in case of the versioned storefile approach change, there is a chance that there are reference files which are not yet archived after compaction because of the change in the store file management. Hence it requires us to add new APIs which ensures that a split can go ahead even if reference files are present but they are in the COMPACTED state.

Current approach



As shown in the above diagram, a scan is operating on 4 files F1, F2, F3 and F4, while a compaction happens on the files F1, F2 and F3 which creates a new file F5, we can see that the HStore does a scanner reset such that the ongoing scan now uses the files F5 and F4. This operation happens under the lock held by the StoreScanner. Since the files F1 to F3 are compacted they are evicted from the block cache and the new blocks of F5 needs to be fetched from the HDFS.

Versioned store file approach



In the above diagram, it can be seen that a scan operation on the files F1, F2, F3 and F4 before compaction, still continues to operate on the same files F1, F2, F3 and F4 though F5 is a newly compacted file but the scanner does not make use of that new file F5. When a new scan is created 'scan2', it can be seen that only the files F4 and F5 are used whereas the files F1 to F3 are not included since they are already compacted.

Once the Cleaner thread runs in the back ground it can be seen that the compacted files from F1 to F3 are removed from the store files list itself.

Note : The C indicates the COMPACTED state on those files that are compacted