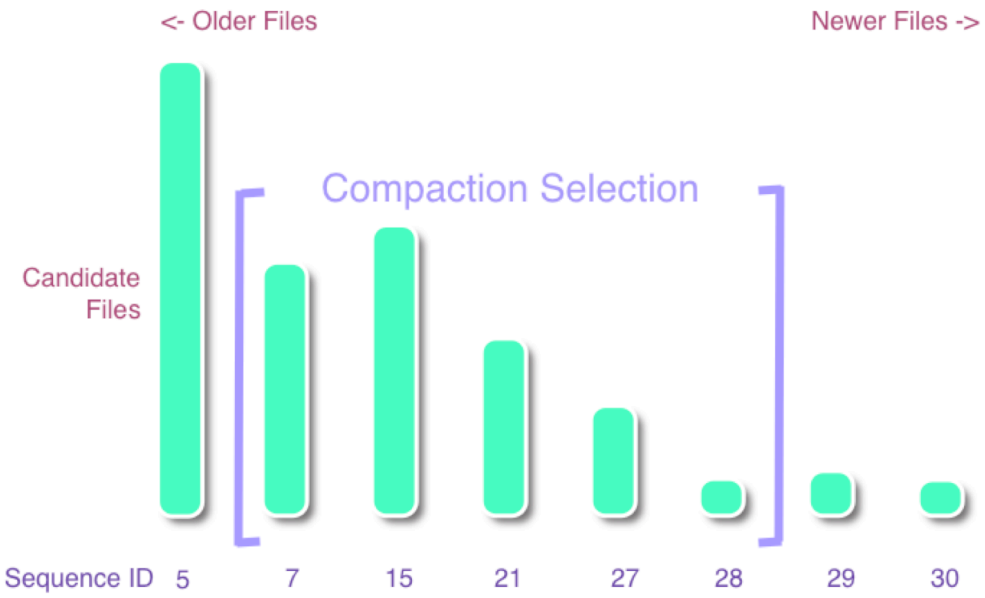
HBase Tier Based Compaction

by Akashnil Dutta

1. Overview

The goal of the compaction selection algorithm is to schedule compactions efficiently. The current algorithm takes a set of candidate files as input, and produces a subset as output. If there is no eligible compactions, the output set can be empty. The candidate set is made of all the files in one region which are not already scheduled for another compaction.

Each file has a special property, called 'sequence id' which indicates the relative recentness of the data in them. Smaller sequence ID implies an older file and vice versa. One additional constraint on the compaction algorithm is that the output subset must have a consecutive range according to sequence ID.*



It is expected that older files are larger, however this is not a requirement. For convenience, the files are kept sorted according to sequenceID. Suppose there are n candidate files, f[0], f[1], ..., f[n-1] from oldest to newest. the algorithm selects a range, [start, end), which specifies the compaction selection {f[start], f[start+1], ..., f[end-1]}. It is important to have

only contiguous sub-sequence of files in a compaction selection because we must maintain a total ordering of the store-files according to recentness, which is used to optimize query performance. Whenever an entry is found in a store-file, we don't need to read any older store-files, because the most recent one must contain the most recent version.

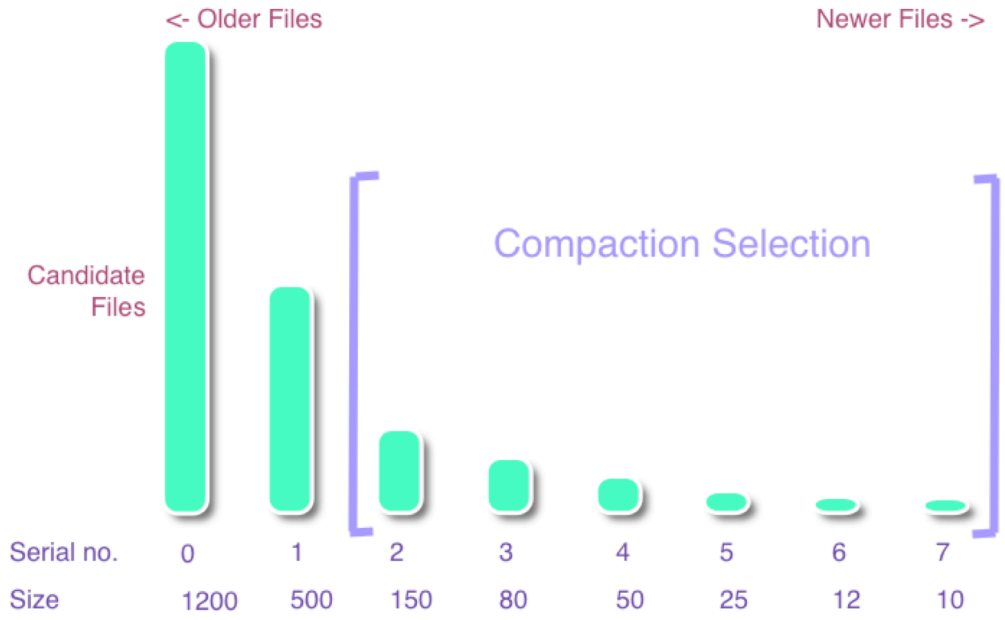
2. Default Compaction Algorithm

The default algorithm always selects a subset consisting of the most recent k files, i.e. in this case end = n always. It picks the smallest value of start which satisfies the following condition:

```
f[start].size <= ratio * (f[start+1].size + ... + f[end-1].size)  // (ratio test)
```

Here ratio is a configurable constant. The default value is usually close to 1.0. Higher values result in more aggressive compactions which keeps the store-file count low but increases the amount of IOPs consumed by compactions.

Suppose the file sizes are given by [1200, 500, 150, 80, 50, 25, 12, 10]. Assume that ratio = 1.0 In this case the subset with the last 6 files are selected, i.e. selection = [150, 80, 50, 20, 12, 10]



If the candidate sizes are [1200, 500, 150, 80, 25, 10], then there is no admissible compactions. Additionally there are several other criteria apart from the ratio test, that needs to be considered for an eligible compaction, which use more configurable parameters.

1. A selection must have a minimum number of files to be eligible.
2. If there is more than a threshold number of files, it is shortened to a smaller one, with upper-bound number of files.
3. No file larger than a threshold is included in any compactions.
4. All files smaller than a threshold is included without the ratio test.
5. Bulk load files are never included in compaction, if configured in that way etc.

3. Tier Based algorithm

The motivation for the tier based algorithm is to provide more fine-grained control, while retaining the efficiency. In the default algorithm, the aggressiveness of the algorithm depends on a single parameter, the ratio. there is no way to control it for different files. In case of the new algorithm, it is possible to set different parameters for different groups of files (based on time-range or size). Old files less likely to be accessed can be given a smaller priority. It is possible to stop compactions for very recent files if cache-on-write is enabled. During every compaction-selection, all the files are assigned to a few different tiers. Next we apply the

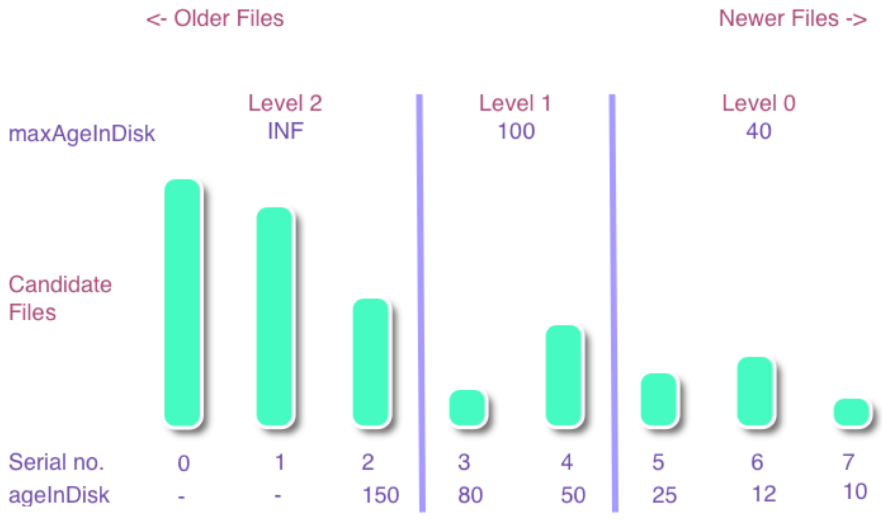
same ratio test for each tier, although with different parameter-values. By default, a compaction selection consists of files from a single tier only. Under special circumstances, there is room for exception (configurable).

3.1 Tier Assignment

There are two ways of categorizing files into tiers. The first one is age-based and the second one is size based.

3.1.1 Age based

In case of default algorithm, we needed only two pieces of information for each file, the sequenceID and the size, for each storefile. For age based assignment, we need to use a time-stamp for the data present in a file. We do not use the use-case specific time-stamps for this purpose, since that can be unreliable. Instead, we add a new metadata field for the minimum flush time for all the data present in a file. In case of a flush file, this value is the time when the file is written to disk. In case of a compacted file, this value is the minimum of the flush-times of its member files. Each tier has a configurable time range. The files are assigned according to this time stamp.

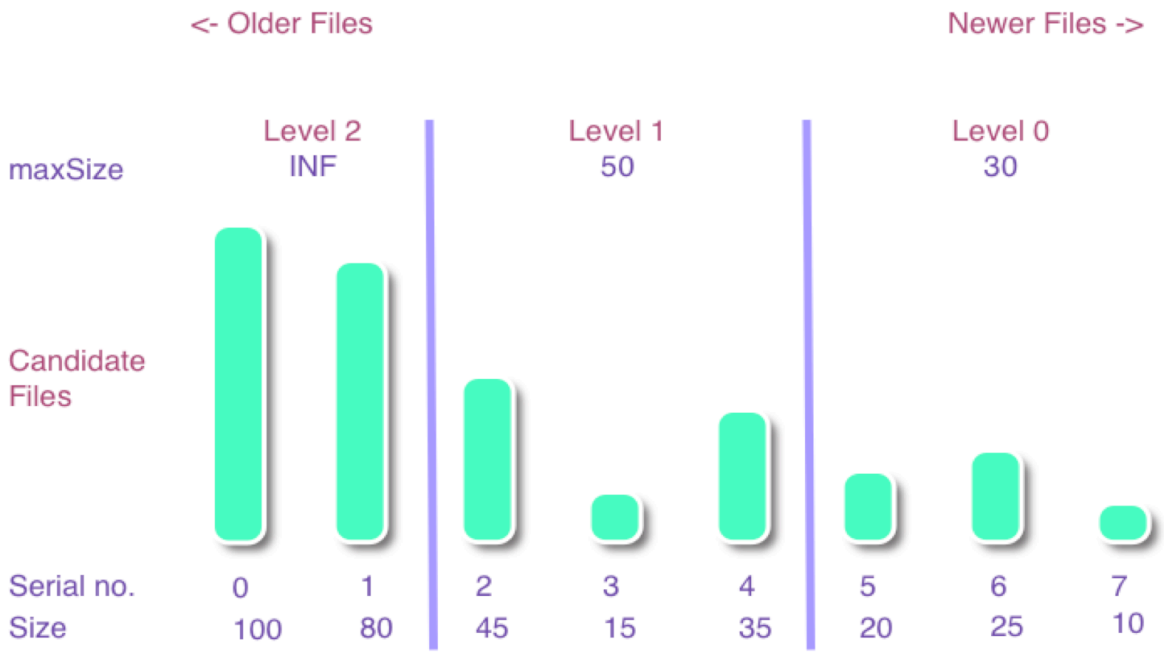


Note: minimum flush time should be strictly increasing with sequence ID. This is a pre-condition for tier assignment. When some of the files don't have minFlushTime field, they are assigned to the highest assigned tier. It is recommended to not use age based tier assignment

immediately following the update to new version, until all files have minFlushTime field, so that files contain the flush time field, before turning on tier based algorithm with timed tier assignment. (See below for online configuration updating)

3.1.2 Size base

Since the file sizes are already available, they can be assigned to tiers according to size ranges. However, tiers must contain consecutive files according to sequence ID. Hence, we make sure that older files always go to higher tiers even if they have smaller size than tier-requirements.



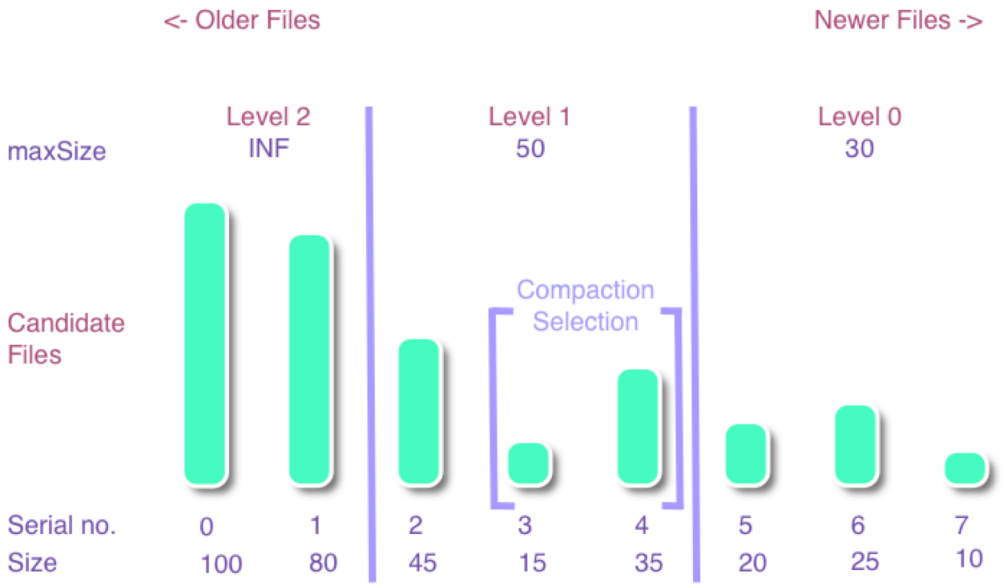
In this example, f[3].size < tier[0].maxSize. However, since f[4].size>tier[0].maxSize, f[3] is assigned to tier[1].

3.2 Compaction Selection

Following the tier assignment, we execute the default selection criteria on each tier, until an admissible selection is found. Suppose tier[i] has files f[u],f[u+1],...,f[v-1]. Then for each pair [start, end) = [i,v) will be tried for i = u,u+1,...,v-1. Every such [start, end) pair will be checked using the ratio test and other criteria until an admissible selection is found. The

order of traversing the tiers is configurable. By default it is set as lower to higher tier. This order can be flipped by setting a Boolean parameter.

For example, consider the tier assignment from the last example. Suppose the ratio for tier 0,1,2 is respectively 0.5,0.5,1.0. Ignore all other checks other than the ratio test. In that case, compactions are admissible in tiers 1 and 2 both. By default, the more recent tier will be selected, i.e., the selection [start, end) = [3, 5) consisting of {f[3], f[4]}.



If the ratio are 1.0,0.4,0.0 for tiers 0,1,2, then selection is admissible in tier 0 only, given by [start, end) = [5,8) consisting of files {f[5], f[6], f[7]}

3.3 Setting parameters

All of the compaction parameters should be located in the file hbase-compactions.xml

The set of all parameters can be different for different column families too. For example, It is possible to use the default algorithm on one column family, the tier based algorithm with one set of parameters for another, and a different set of parameters on a third one. The general template for each parameter is of the form

"hbase.hstore.compaction.**$schema**.**$attribute**"

The schema can be specified in the form "tbl.$TableName.cf.$FamilyName" or as the string "default". If the settings for some column family is not specified individually, the default settings are used in that case.

There are two kinds of parameters for the algorithm, the tier specific parameters, and the tier independent parameters. Tier independent parameters are set for the overall algorithm. On the other hand, tier specific parameters can be different for each tier and are used while executing the subroutine for a particular tier only.

For a tier independent parameter, "$schema" is just the parameter name. For a tier specific parameter, "$schema" is of the form "tier.$num.$parameterName". A tier specific parameter can be set without specifying the tier. Then that value will be used as a default value for any tier for which that parameter has not been set explicitly.

For example, NumCompactionTiers, MinCompactSize are tier independent parameters. It can be set in the following way"

"hbase.hstore.compaction.default.MinCompactSize"
"hbase.hstore.compaction.tbl.table1.cf.family1.MinCompactSize"

If not set for family2, the default value is used for that case. CompactionRatio is a tier specific parameter. We can set parameters by using all of the following keys:

"hbase.hstore.compaction.default.CompactionRatio"
"hbase.hstore.compaction.tbl.table1.cf.family1.CompactionRatio"
"hbase.hstore.compaction.tbl.table1.cf.family1.tier.2.CompactionRatio"

In this case, the tier 2 for the store corresponding to tbl.table1.cf.family1 will use the third parameter, for all other tiers in that store, the second parameter will be used. If there are other column-families for which no parameter has been set, the first value will be used for them.

3.3.1 Tier independent compaction paramenters

Parameter	Description	Default	Comments/Caution
CompactionPolicy	The policy to be used, default or tier-based	DefaultCompactionPolicy	Must be a valid class name
MaxCompactSize	Upper bound on the file size in minor compaction, in bytes	INF	Set high enough value
MinCompactSize	Lower bound below which every file is compacted, in bytes	0	Set small enough
ShouldExcludeBulk	Whether bulk load files should be excluded from minor compactions	default*	Make sure bulk load files are the oldest ones, for now
ShouldDeleteExpired	Whether TTL-expired files should be first deleted by compaction	default	Set as true
ThrottlePoint	The threshold size for assigning compactionSelections into the small/large queue, in bytes	default	Too large or too small will starve one of the queues
MajorCompactionPeriod	The interval at which major compactions are selected periodically, in milliseconds	default	keep long enough
MajorCompactionJitter	The fractional deviation from MajorCompactionPeriod which is randomly effected to avoid having major compactions in many stores at the same time.	default	default is good
NumCompactionTiers	The number of tiers to assign storeFiles in	1	Careful to set this correct
IsRecentFirstOrder	Whether the tiers will be tried for eligible compactions from newest to oldest	true	Set as false if mixing tiers. Beware of starvation of small compactions.

3.3.2 Tier specific parameters

Parameter	Description	Default	Comments/Caution
MaxAgeInDisk	The maximum age of a file which can belong to this tier, in milliseconds. tier[i] contains all files with age between (tier[i-1].maxAgeInDisk, tier[i].maxAgeInDisk]	INF	Set this carefully (No default value). should be increasing from lowest to highest tier.
MaxSize	The maximum size of a file which can belong to this tier, in bytes. tier[i] contains all files with size between (tier[i-1].maxSize, tier[i].maxSize]	INF	Usually use only one of MaxAgeInDisk and MaxSize criteria for tier assignment. However it is possible to use both.
CompactionRatio	The parameter used for ratio test in this tier	default	The most important parameter for each tier. Set as zero to block all compactions for this tier. Set higher for more aggressive compactions.
MinFilesToCompact	Minimum number of files to be in a selection in this tier	default	An important parameter. Should be at least 2.
MaxFilesToCompact	Maximum number of files to be in a selection in this tier	default	Setting this small will result in more IOPs in long term, but finish compactions quicker.
EndInclusionTier	This is a special feature which allows compaction selection across more than one tier. Suppose tier[i].endInTier = j, tier[i] contains filesf[u],f[u+1,...,]f[v], and tier[v] contains files upto f[w]. Then all pairs [start, end) = [i,w) will be tried for i = u,u+1,...,v-1 using the parameter-set for this tier.	this.tierIndex	Be careful when setting this parameter. Following condition must be true:i>=tier[i].endInclusionTier>=tier[i-1].endInclusionTier>=0 for all i>0 (The second inequality is important for avoiding not-in-order compactions).

Default indicates the parameter from the default compaction algorithm. Even if ShouldExcludeBulk is not set for the tier based compaction algorithm, if tier based algorithm is turned on, it will use the parameter from the default algorithm.

3.3.3 Online updating of configuration

It'll be possible to update the configuration without restarting. An RPC call will be sent from the client. All of the parameters will be reloaded from the files into the compound configuration object. The compaction related ones will take effect. However it is recommended not to change any other parameters before updating, as it may result in inconsistent behavior and local caching problems.

hbase-trunk/src/main/java/org/apache/hadoop/hbase/util/UpdateConfigTool

It will be possible to update only one region-server by passing an argument.