Using stripe compactions

sershe, HBASE-7667, 2013-04-26

“Stripe compactions” is a scheme that splits the data inside the region by row-key, creating sub-ranges of data invisible from outside. This improves read performance in common scenarios and greatly reduces variability, by avoiding large and/or inefficient compactions. For details, see design doc attached to HBASE-7667.

This document describes how to configure and use this improvement.

Table of Contents

Use cases and considerations **1**

Enabling and configuring **2**

How to configure compactions for individual tables or CFs 2

Enabling stripe compactions 2

Tuning: important settings 2

Other settings 3

Use cases and considerations

There are two primary use cases for stripe compactions. Given that, it may make sense to enable stripe compactions on individual tables, or even column families, not on the entire cluster. The cases are as such:

1. Approximately uniform keys and large regions.
   Large depends on your load, network and server configuration, but generally you should not use stripe compactions for uniform data if your regions are less than 2Gb in size.
2. Non-uniform data with certain row-key patterns (for example, sequential logs).
   Multiple policies can be implemented that can only compact a subset of files in the necessary key ranges. Currently, one is implemented for sharded, roughly sequential keys (it will work with other data but will be suboptimal).

Additionally, if you often perform very wide scans (spanning entire region/regions), you should verify the performance – stripe scheme uses larger number of files than default to ensure all compactions are small, which can affect very wide scans. It doesn't affect gets or narrow scans, in fact it may make them faster.

Also, there is no such thing as "major compaction" in the stripe scheme, so scheduling one will merely cause a minor compaction to be scheduled.

Enabling and configuring

How to configure compactions for individual tables or CFs

All the below configuration parameters can be set in hbase-site.xml; they can also be set for a table using the following syntax in the shell:

```
alter <table>, CONFIGURATION => {<setting> => <value>}
```

or for a column family, using the following syntax:

```
alter <table>, {NAME => <column family>, CONFIGURATION => {<setting> => <value>}}
```

For example:

```
alter 'orders', CONFIGURATION => {' hbase.store.stripe.fixed.count' => 10}
```

```
alter 'logs', {NAME => 'blobs', CONFIGURATION => {'hbase.store.stripe.fixed.count' => 10}}
```

Enabling stripe compactions

The following settings need to be set to use stripe compactions.

Setting	Reasonable value	Comment
hbase.hstore.engine.class	org.apache.hadoop.hbase.regionserver.StripeStoreEngine	Enables the engine.
hbase.hstore.blockingStoreFiles	14–200	Stripe scheme uses more files than the default scheme, so if this is not set, the store will be blocked often. Always set this when enabling stripes. Rule of thumb: • for uniform scheme, 7*(# of stripes, see Tuning); • for size scheme, 14+2*(max # of stripes, see Tuning), or a high value (200, for example).
hbase.store.stripe.sizeBased	true, or default	By default, fixed-count scheme is used, which is good for uniform data. This setting enables the custom size-based scheme.

Tuning: important settings

The most important setting to decide for uniform data is the number of stripes. For sequential data, it is the size of the "latests" stripe after which it will be split. For the rest of the settings, defaults work reasonably well, but you can consider tuning them.

Setting	Reasonable value	Comment
Uniform data		
hbase.store.stripe.fixed.count	4–30	The number of stripes to use. Small number of stripes. **If too low**, there will be less positive effect (bigger compactions). In fact, setting 2-3 stripes can be a net negative (there's some overhead for little benefit over default scheme). Large number of stripe can be better, especially for reads - the compactions will become smaller and more numerous. **If too high**, there will be too many files open and tiny compactions. So: • start within 6-12 range - evaluate performance if setting higher or lower;

		• if you know your region size, target 500Mb-1Gb stripes (e.g. for 5Gb region 6 is a good starting point, for 20Gb region – 12+ is a good starting point).
hbase.server.compactchecker.interval.multiplier	1-3, or default	Only necessary if you have many (>15) stripes and lots of writes. In this case, there are a larger number of smaller compactions, so additional opportunities are needed to schedule them; compaction checker should run more often. Note that this is *checker* frequency, not compaction frequency. To have it run every 10-30sec., set to 1-3 (assuming you have the default hbase.server.thread.wakefrequency setting of 10sec.). Adjust accordingly if you changed that frequency. **If too high**, too many files may accumulate, slowing reads.
hbase.store.stripe.fixed.initialminfilesL0	3-15, or default	The number of files (flushes and bulk loads) to accumulate initially, before determining how to stripe the data. Think about it as region splitting – how soon is there enough data to find good region boundary? Set depending on how uniform your keys are. Default is 8. **If too low**, boundaries may be suboptimal if data in the first files doesn't represent the later data well. **If too high**, initially you will have many files in the region, which can cause slow reads. No effect later on.
Sequential data		
hbase.store.stripe.size.split.at	500Mb-5Gb, or default	TBD
General		
hbase.store.stripe.compaction.minFilesL0	3-7, or default	The number of files shared between stripes to accumulate before data in them is striped. Default is 4. **If too low**, there will be many unnecessary compactions. **If too high**, there may be files necessary for reads, which can slow them down.
hbase.store.stripe.compaction.ratio, hbase.store.stripe.compaction.maxFiles, hbase.store.stripe.compaction.minFiles	1.1-1.4, 7+, 2-4; or defaults	Same as ratio, min files, and max files settings for default compactions. Used when compacting the files inside stripes. Same guidelines apply as to default compactions. Due to lower number of files in each stripe than in a normal region, though, you might want to not set minFiles higher than 4.

Other settings

These settings should almost always be good with default values.

Setting	Reasonable value	Comment
Uniform data		
hbase.store.stripe.fixed.rebalance.at	1.5+, or default	The skew between some stripe size and the average that will prompt rebalancing of the stripes. Rebalancing is a relatively large compaction (several entire stripes), so don't set this until you can pinpoint the skew inside regions as a source of some problems. Default 3.0.
hbase.store.stripe.fixed.maxrebalance	2-5, or default	The maximum number of stripes to rebalance in single compaction if the above rebalance is triggered. Size of rebalance compaction will be roughly ((this setting)+(above setting)-1)*(region size)/(number of stripes). Set accordingly depending on number of stripes you use. **If low**, more compactions may be needed to rebalance, **if high**, there's a risk of large compaction. Default is 2.

Sequential data		
hbase.store.stripe.size.split.fraction	0.7-0.9, or default	TBD
hbase.store.stripe.size.maxstripes	20-100, or default	TBD
hbase.store.stripe.size.maxmerge	3-10, or default	TBD
General		
hbase.store.stripe.split.max.imbalance	1.1+, or default	Set to lower value if you see abnormal asymmetrical region splits. During region split, by default the stripe scheme tries to split region on the stripe boundary. If this results in too much skew between two sides of the split (more skew than this setting), the value in the middle of a stripe is chosen. Currently, choosing stripe boundary value has no advantage, but it may be used for future improvements for splits. Default is 1.5.
hbase.store.stripe.compaction.assume.ordering	true, or default	This can be used to reduce compaction amount somewhat, if you don't use out-of-order puts and deletes (i.e. don't set custom timestamps on puts and deletes). There's a problem in HBase described in "Deletes mask puts" (http://hbase.apache.org/book.html#d2520e3590). This setting will make time window for this problem, where query results change, wider. It shouldn't have much effect if you don't use custom timestamps. By default this is disabled.