Contents

[HBASE compaction simulator 2](#_Toc379804654)

[1.0 Introduction 2](#_Toc379804655)

[2.0 Configurations 2](#_Toc379804656)

[3.0 Titan 7](#_Toc379804657)

[3.1 Enabling/disabling major compactions 7](#_Toc379804658)

[3.2 Changing compactionMinFiles 8](#_Toc379804659)

[3.3 Changing compaction ratio 10](#_Toc379804660)

[4.0 ODS 12](#_Toc379804661)

[4.1 Enabling/disabling major compactions 12](#_Toc379804662)

[4.2 Changing compaction ratio 13](#_Toc379804663)

[4.3 Changing compactionMinFiles 16](#_Toc379804664)

[5.0 Testing different compaction algorithms 17](#_Toc379804665)

[6.0 Configurations analysis 22](#_Toc379804666)

[6.1 Compaction ratio analysis 22](#_Toc379804667)

[6.2 CompactionMinFiles analysis 30](#_Toc379804668)

[6.3 CompactionMaxFiles analysis 33](#_Toc379804669)

[6.4 Gathering everything together 36](#_Toc379804670)

[7.0 Editing Simulator 37](#_Toc379804671)

[7.1 Adding additional general configuration 37](#_Toc379804672)

[7.2 Adding additional compaction algorithm 38](#_Toc379804673)

[7.3 Adding additional compaction configuration 38](#_Toc379804674)

# HBASE compaction simulator

## 1.0 Introduction

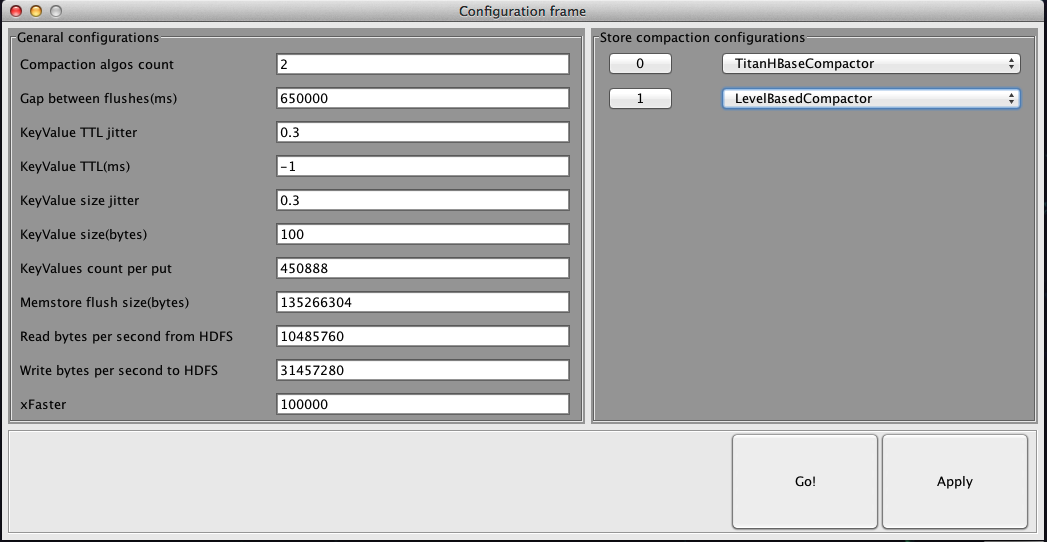
Compaction simulator simulates HBase work. As input it gets configuration of HBase and what compaction algorithms and its confs you want to see. As output it provides Write Amplification Factor (WAF) graph and Read Amplification Factor (RAF) graph. Basically WAF = bytes written and read from disk during compactions / bytes written to disk during flushes, so WAF shows how many times we write the same data again and again to disk. The WAF is a significant factor in write performance and an algorithm with a WAF that is 10X higher is likely to support 10X fewer writes/second (copied from wiki page by Mark Callaghan). More information about WAF and RAF is in [[Database/PerformanceModels]] So given configurations of HBase itself and compaction algorithms you want to test, simulator show WAF and RAF for each compaction algorithm, so that you evaluate results and see what configuration are better. Even more, simulator allows user to change some configuration during simulating, for example you can change load and see how graphs are changing with changing load.

P.S. code is submitted to GitHub: *https://github.com/ibragimismailov/EventBasedSimulator*

## 2.0 Configurations

You can change configurations to see the difference in result. Actually main purpose of this simulator was to find better configuration, than we already have for Titan and ODS clusters. There are no special bounds for configuration values, anyway if you will make a typo while setting configurations, simulator will warn you about that.

When you launch simulator application, you see ConfigurationFrame. Configurations on this frame are divided into 2 parts: general HBase configurations and compaction algorithms configurations:

 [[Image:HBase--Compaction\_Simulator--GeneralInformation--finalgeneralConfs.png]]

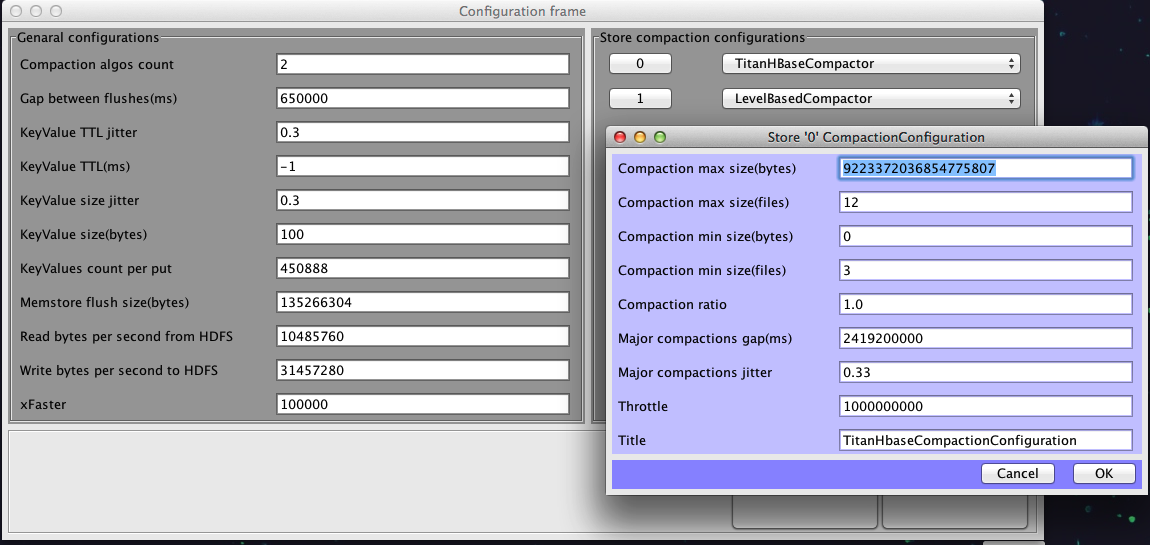
After changing any of this configurations, you need to hit Apply button. So in the left side you can see general HBase configurations like:

* **Gap between flushes(ms)** - time gap between two consecutive flushes in each store;
* **Memstore flush size(bytes)** - size of MemStore;
* **Compaction algos count** - amount of compaction algorithms you want to test - this value is bounded with range [1, 10];
* **KeyValue TTL(ms)** - average TTL for KeyValues in HBase, user must enter "-1" if there is no TTLs for KeyValues (otherwise all TTL info about KeyValues is stored - it is very bad to store it if we use it to delete expired KeyValues) - this value is bounded with range [-1, 100 days];
* **KeyValue TTL jitter** - jitter used for calculate TTL of each KeyValue - this value is bounded with range [0, 1]. TTL of each KeyValue is a random value in range [KeyValue TTL(ms)\*(1.0-jitter), KeyValue TTL(ms)\*(1.0+jitter)];
* **KeyValue size(bytes)** - average size of KeyValue in HBase;
* **KeyValue size jitter** - jitter used for calculating KeyValue size - this value is bounded with range [0, 1]. Size of each KeyValue is random value in range [KeyValue size(bytes)\*(1.0-jitter), KeyValue size(bytes)\*(1.0+jitter)];
* **Read bytes per second to HDFS** - read rate throughput from HDFS;
* **Write bytes per second from HDFS** - write rate throughput to HDFS;
* **xFaster** - speed coefficient. Makes simulator simulate everything xFaster times faster - this value is not bounded, BUT recommended value is (Gap between flushes)\*(Compaction algos count)/5. So for example if xFaster = 10000, that means that simulator will simulate 1 year in 50 minutes
* **kvsPerPut** - size of KeyValuePack. KeyValuePack is some amount of KeyValues that we send to HBase as one object during put, considering as we are sending a bunch of KeyValues

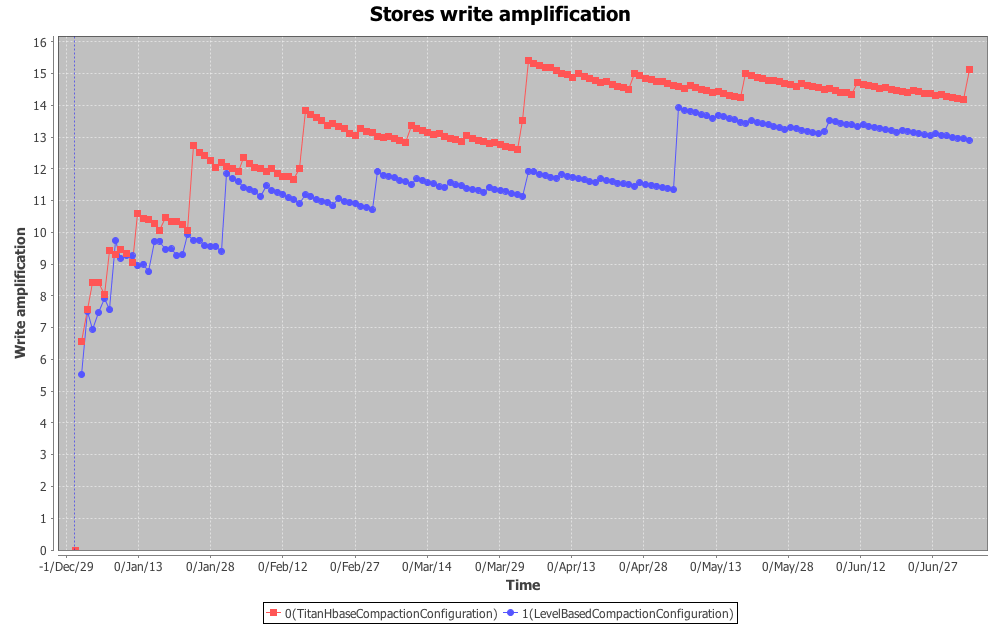
Changing "Compaction algos count" field you can have different number of compaction algorithms to compare. So in the right side of this frame you have compaction specific configurations - for each store you have separate compaction algorithm with separate configurations. There is a list of configurations:

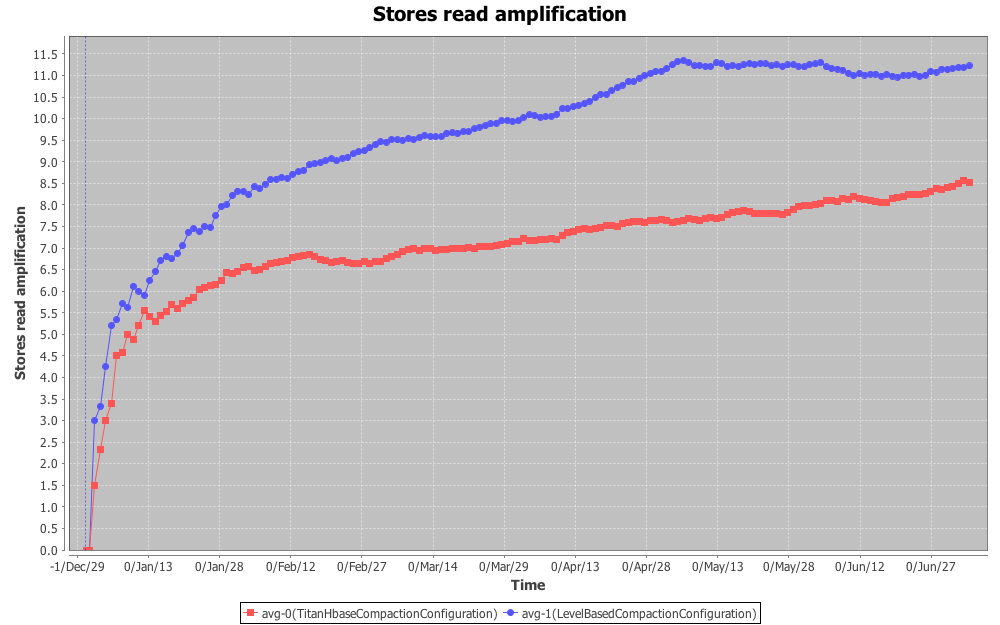
* **HBaseCompactor** - it is HBase native compaction algorithm with default values of configurations;
* **TitanHBaseCompactor** - it is HBase native compaction algorithm with configuration, used in Titan cluster;
* **ODSHBaseCompactor** - it is HBase native compaction algorithm with configuration, used in ODS cluster;
* **LevelBasedCompactor** - it is level-based compaction algorithm. For clarification see comments and descriptions of classes' and methods' in code;
* **IbraCompactor** - some algorithm, created by myself. For clarification see comments and descriptions of classes' and methods' in code;
* **RandomCompactor** - just very simple random-based compaction algorithm.
* **RandomHBaseCompactor** - it is HBase native compaction algorithm with random values of configurations;

so, after selecting compaction algorithm type, you can change some configurations for this algorithm. To do that you need to hit the corresponding button:

 [[Image:HBase--Compaction\_Simulator--GeneralInformation--finalcompactionConfs.png]]

When you finish editing configuration, hit button “Go” to see Write amplification graph and Read amplification graph. X-axis of these charts is time in format (year/month/day). For example here you can see comparison of TitanHBaseCompactor and LevelBasedCompactor:

 [[Image:HBase--Compaction\_Simulator--GeneralInformation--finalSWAexmaple.png]]

 [[Image:HBase--Compaction\_Simulator--GeneralInformation--finalSRAexmaple.png]]

## 3.0 Titan

So, if we just do simulation with HBase compaction simulator with Titan configuration, so here are the result graphs, that we get:

### 3.1 Enabling/disabling force major compactions

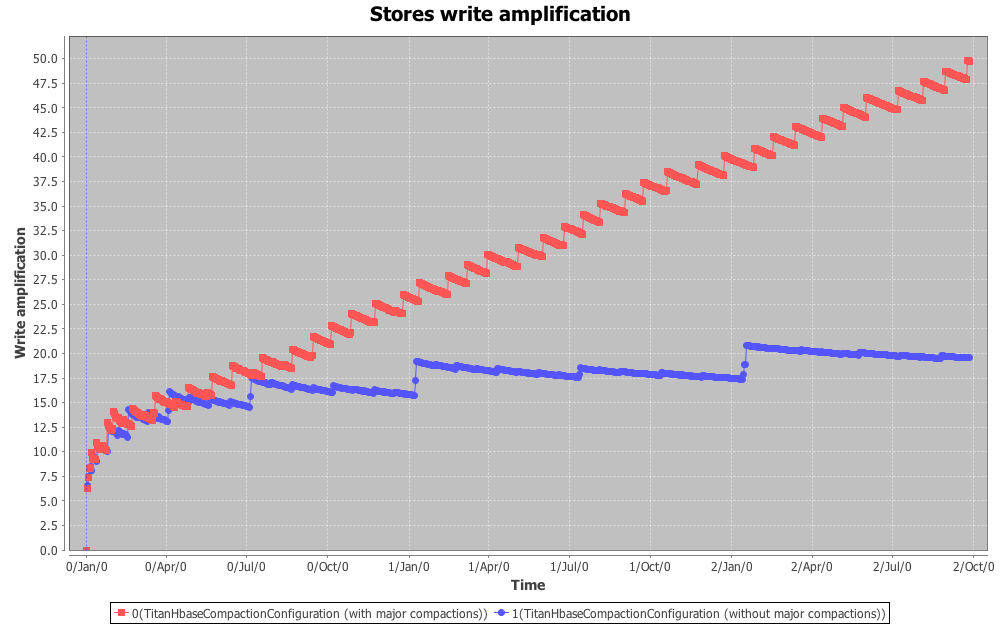
As You can see from next charts, force major compaction makes WAF increase linearly, while graph without force major compactions is logarithmic:

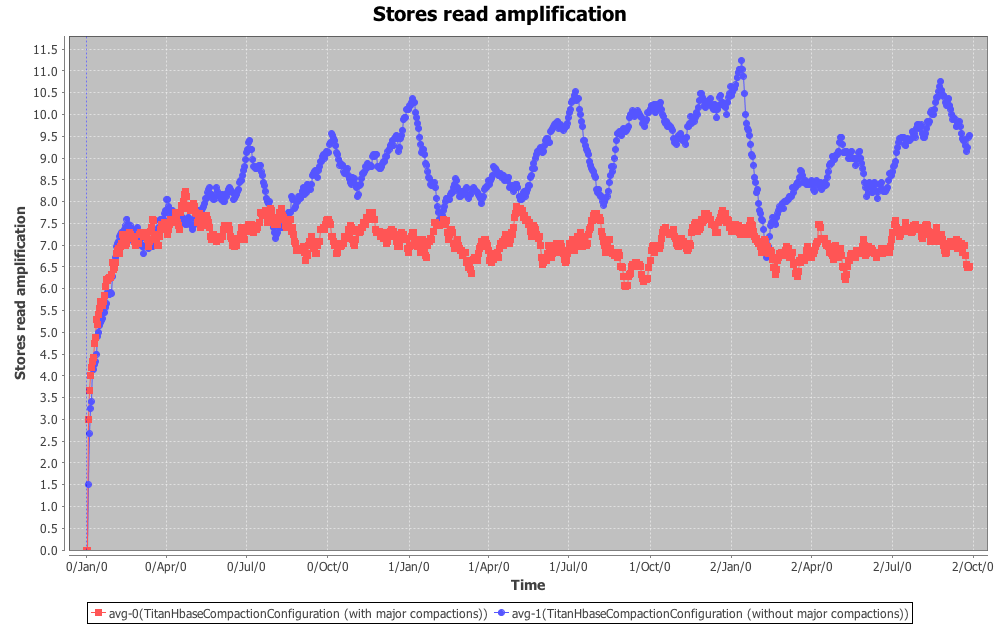
* **with force major compactions**: WAF = 1.13\*m+11.9
* **without force major compactions**: WAF = 2.21\*log(225.19\*m)

where m - is amount of months passed.

On the other hand, RAF chart show that using force major compactions makes RAF lower (it improves performance)

* **with force major compactions**: RAF = 7;
* **without force major compactions**: RAF = 9;

 [[Image:HBase--Compaction\_Simulator--Titan--finalmajorCompareSWAsmall.png]]

 [[Image:HBase--Compaction\_Simulator--Titan--finalmajorCompareSRAsmall.png]]

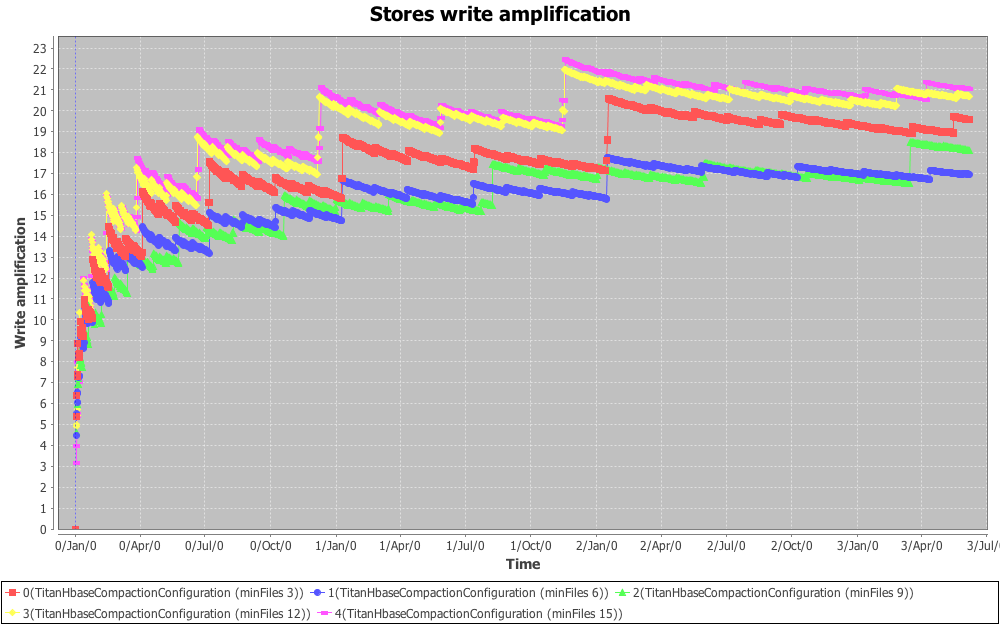
So it can be reasonable to disable force major compactions, or at least make them happen rarely.

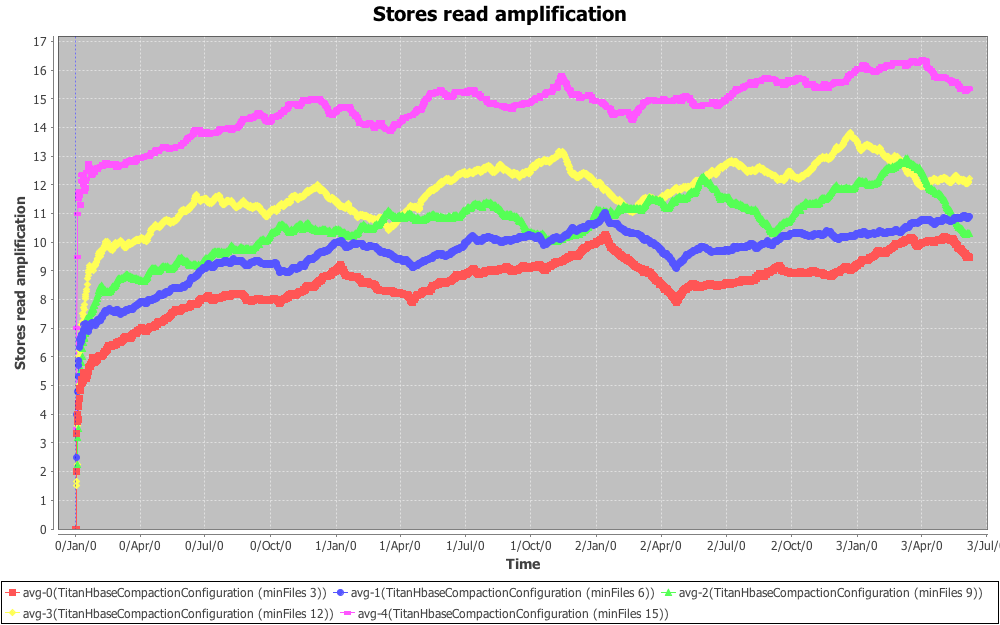
### 3.2 Changing compactionMinFiles

We run simulations of TitanHBaseCompactor with different values of compactionMinFiles (3, 6, 9, 12, 15) - default value is 3. Analyzing these charts, we can get the following formulaes:

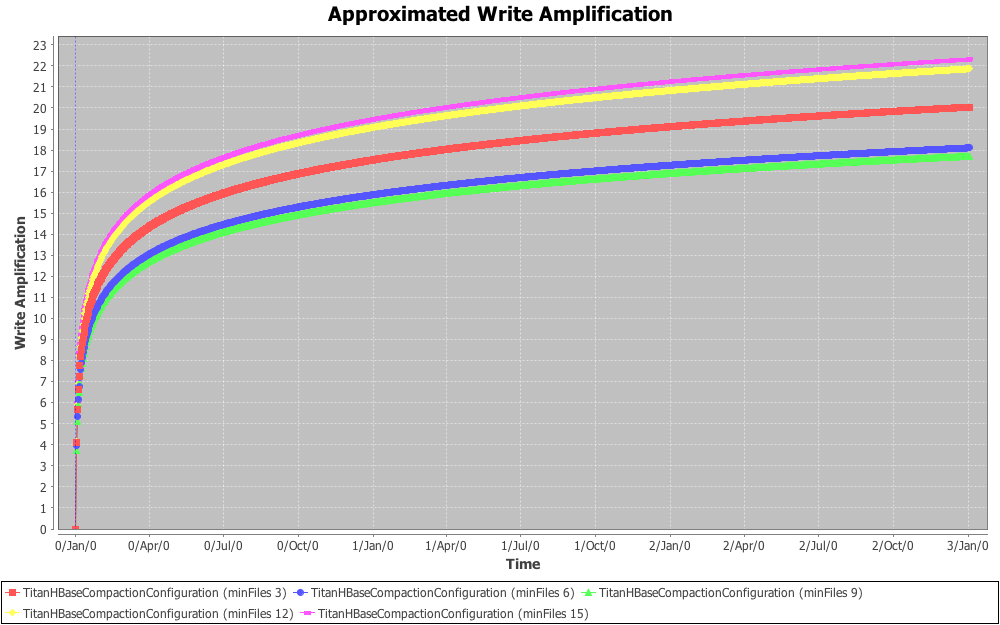
* compactionMinFiles=3) SWA = 2.27548 \* Math.log(182.442\*m);
* compactionMinFiles=6) SWA = 2.02147 \* Math.log(212.009\*m);
* compactionMinFiles=9) SWA = 1.99621 \* Math.log(194.037\*m);
* compactionMinFiles=12) SWA = 2.52863 \* Math.log(155.267\*m);
* compactionMinFiles=15) SWA = 2.59043 \* Math.log(149.980\*m);

where m - is amount of passed months.

 [[Image:HBase--Compaction\_Simulator--Titan--finalminFilesSWAsmall.png]]

 [[Image:HBase--Compaction\_Simulator--Titan--finalminFilesSRAsmall.png]]

So we can approximate graphs like this:

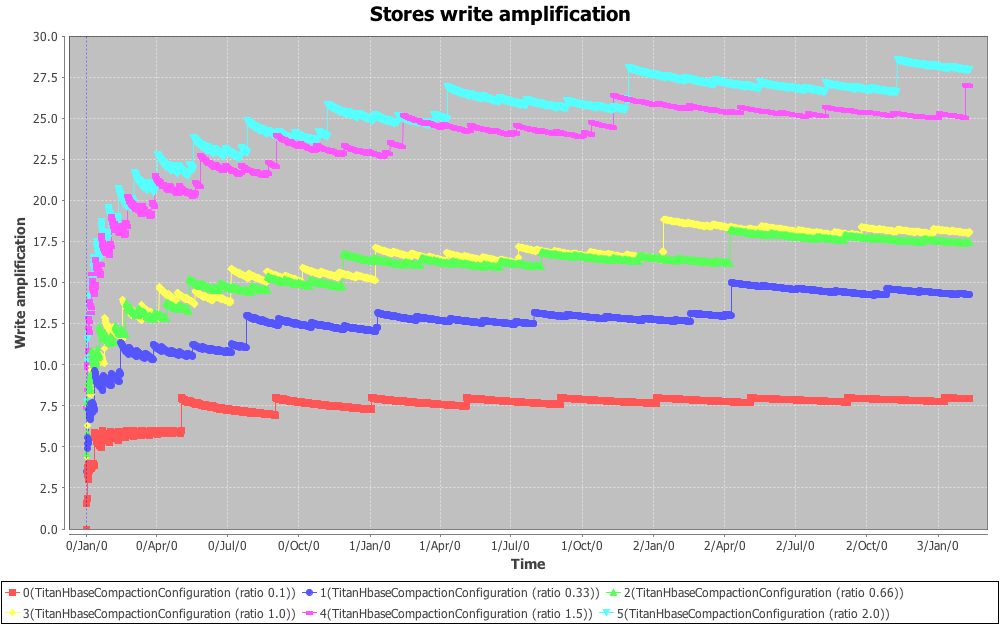
 [[Image:HBase--Compaction\_Simulator--Titan--finalaproxSWA.png]]

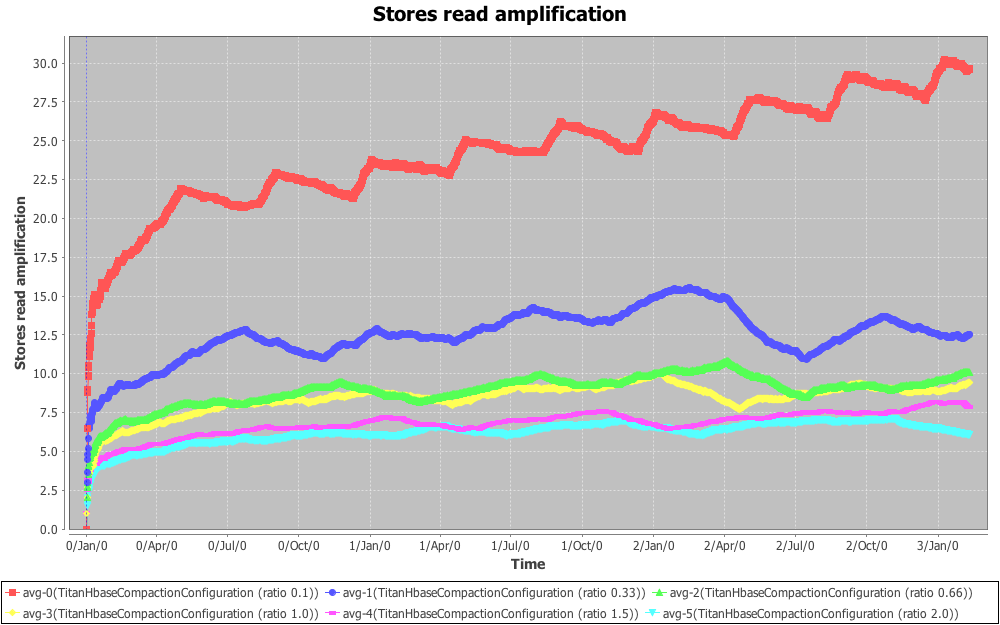
So choosing compactionMinFiles as 6, we can decrease WAF by 2.0, but RAF will increase by 1. If it makes sense for Titan cluster, probably we can do that.

### 3.3 Changing compaction ratio

We run simulations of TitanHBaseCompactor with different values of compactionRatio (0.1, 0.33, 0.66, 1.0, 1.5, 2.0) - default value is 1.0. Analyzing these charts, we can come up to the following conclusion - you really can choose another value for ratio, than 1.0 (which is currently used) - for example 0.33 or 0.66 or something between them:

* ratio=0.33) decreases WAF by 3.5 and increases RAF by 3.5

 [[Image:HBase--Compaction\_Simulator--Titan--finalratioSWAsmall.png]]

 [[Image:HBase--Compaction\_Simulator--Titan--finalratioSRAsmall.png]]

## 4.0 ODS

So, if we just do simulation with HBase compaction simulator with ODS configuration:

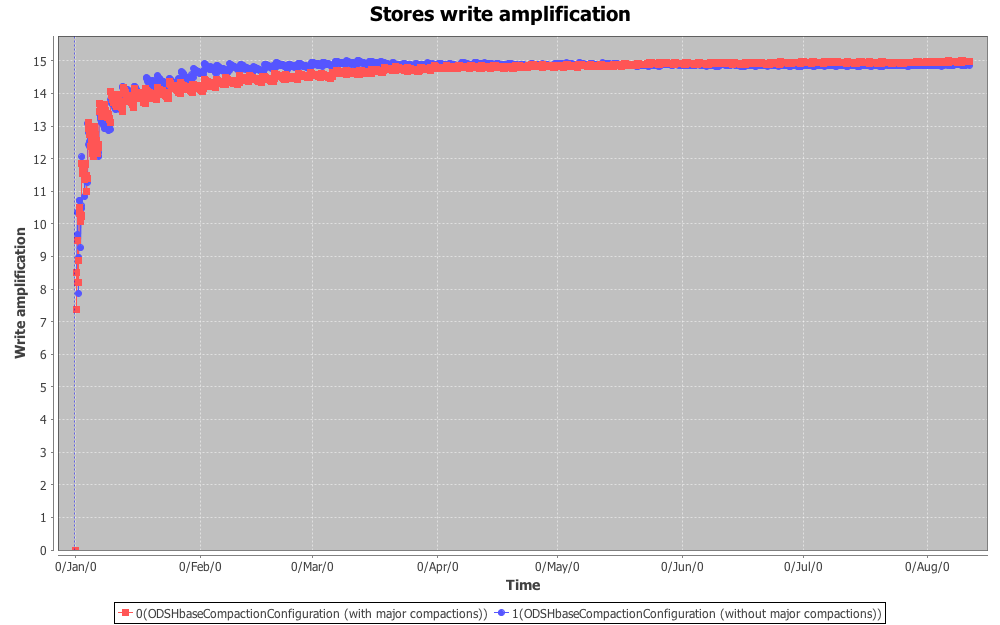
### 4.1 Enabling/disabling force major compactions

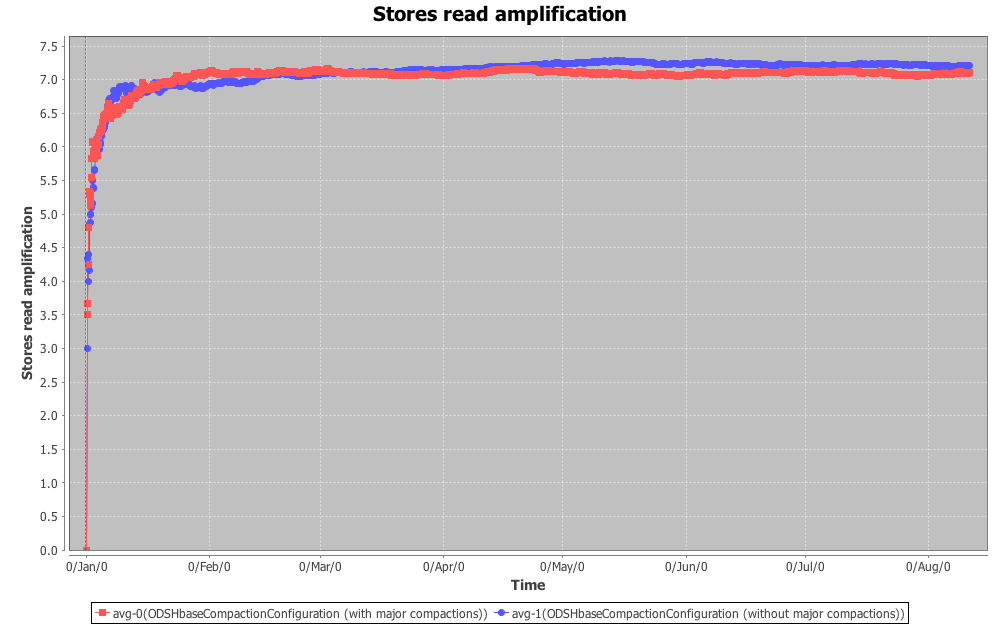
It is obvious that due to small KeyValue TTL in ODS configuration (3 days), WAF and RAF becomes steady very quickly.

WAF = 15;

RAF = 7.2

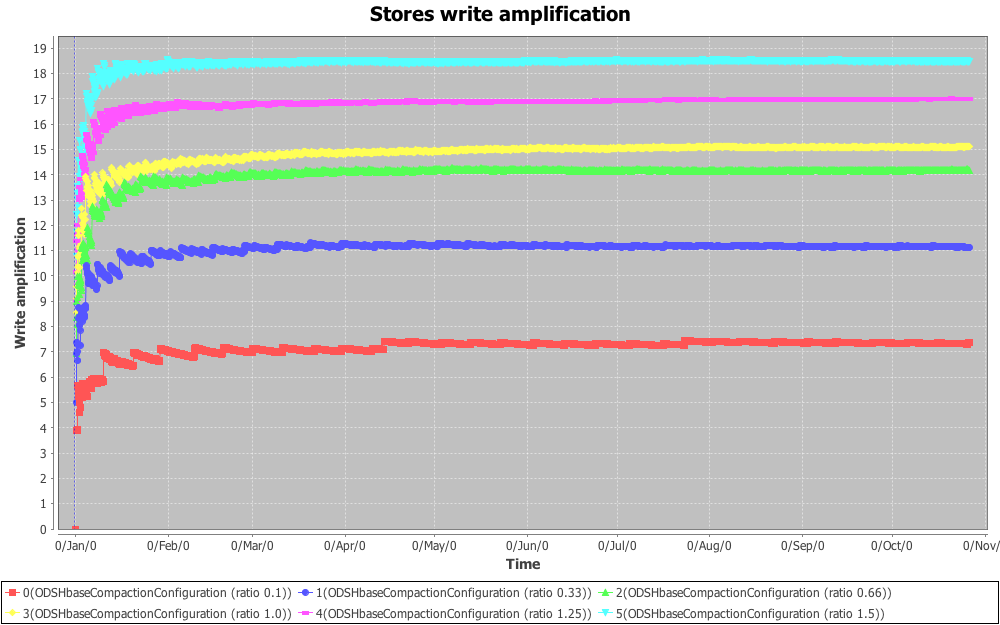
Also you can see that due to small TTL enabling/disabling force major compactions doesn't make any difference. So as far as force major compactions doesn't make any difference and force major compactions doesn't do anything that can't be done during minor compactions, why do we have it?

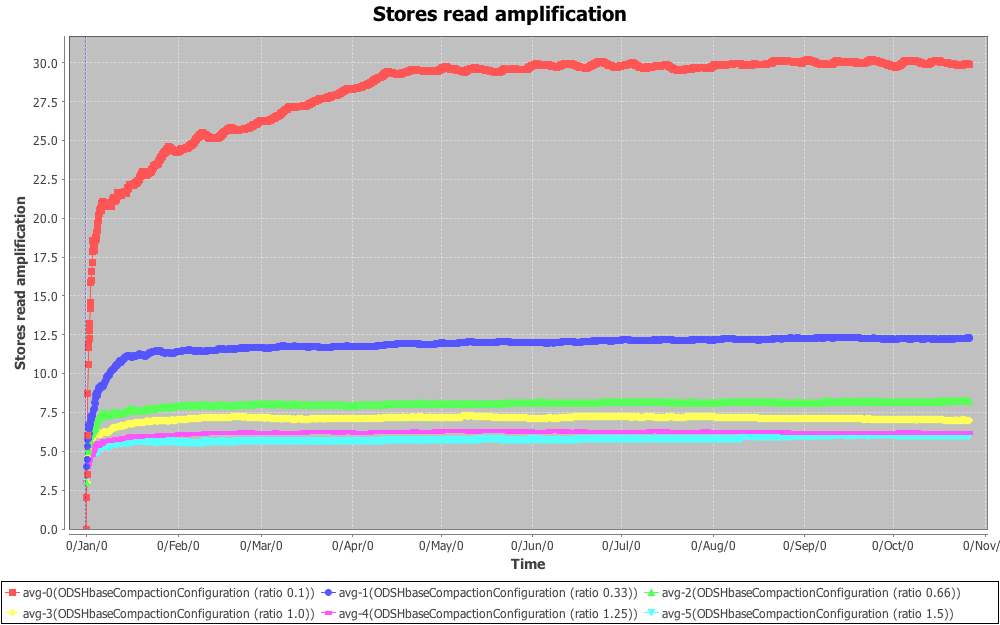
 [[Image:HBase--Compaction\_Simulator--ODS--finalmajorSWAsmall.png]]

 [[Image:HBase--Compaction\_Simulator--ODS--finalmajorSRAsmall.png]]

### 4.2 Changing compaction ratio

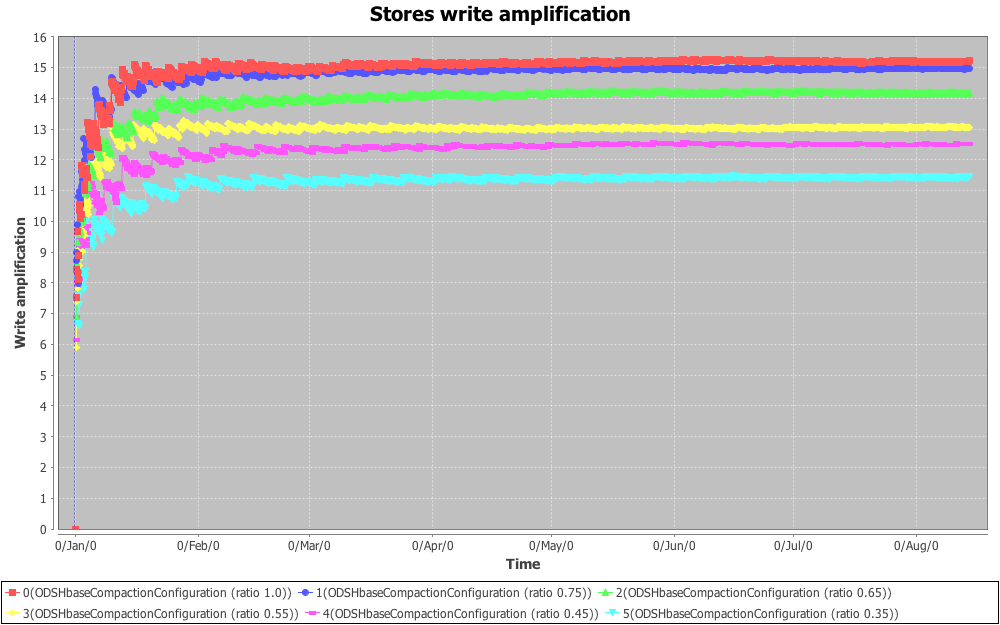
So changing compaction ratio, we can see that to to decrease WAF, we need to choose ratio < 1.0

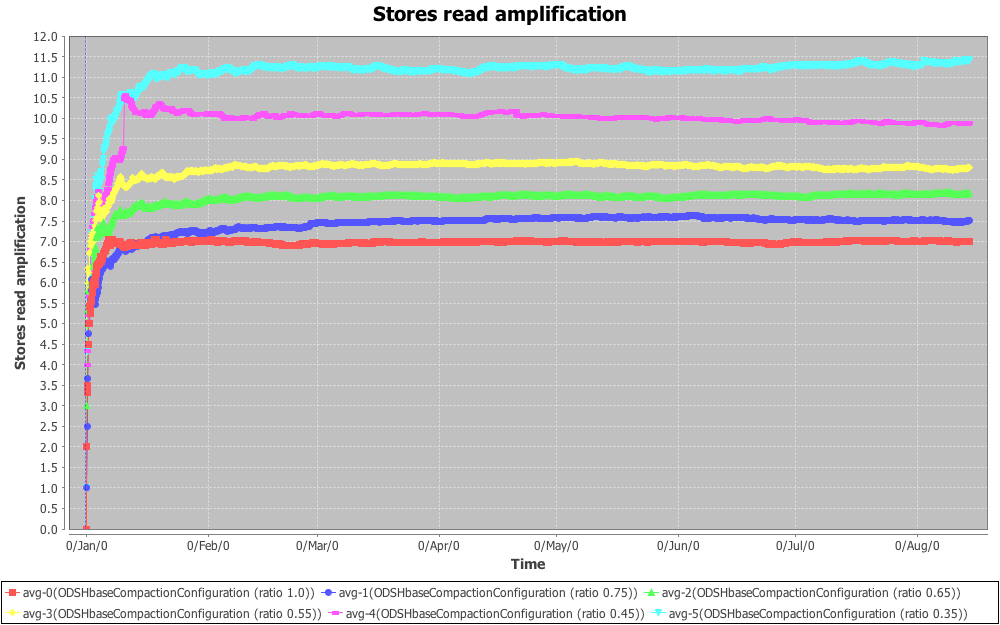
 [[Image:HBase--Compaction\_Simulator--ODS--finalratioSWAsmall.png]]



[[Image:HBase--Compaction\_Simulator--ODS--finalratioSRAsmall.png]]

, so lets take a little closer look at graphs with ratios {0.75, 0.65, 0.55, 0.45, 0.35}

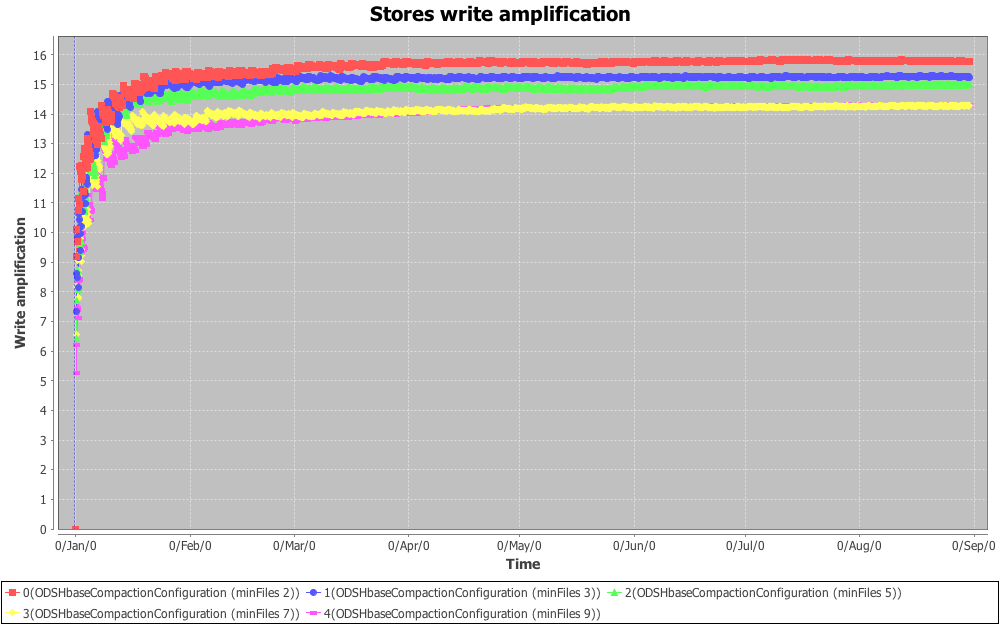
 [[Image:HBase--Compaction\_Simulator--ODS--finalratio2SWAsmall.png]]

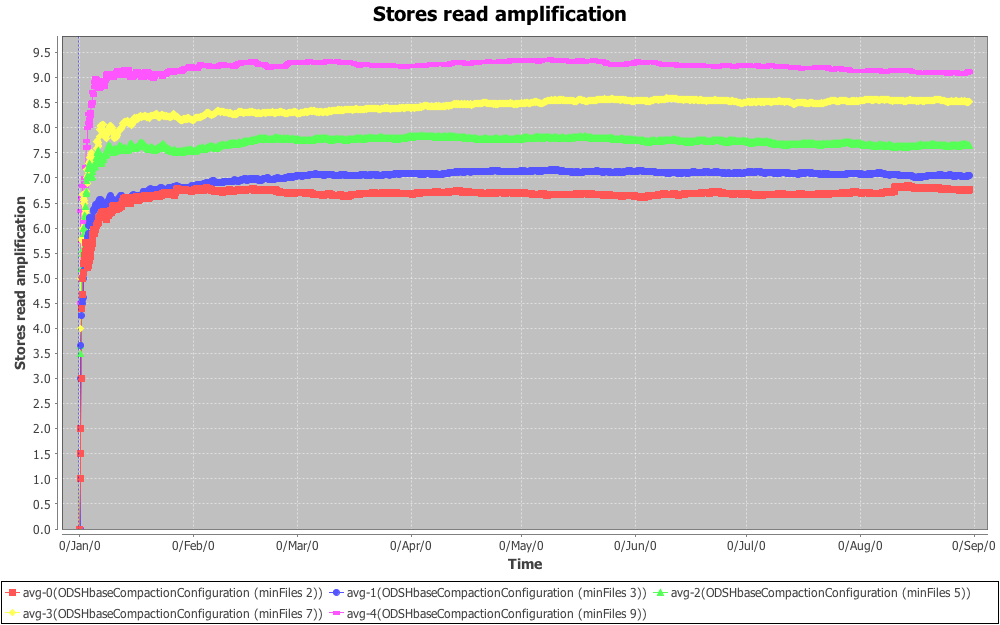
 [[Image:HBase--Compaction\_Simulator--ODS--finalratio2SRAsmall.png]]

So you can see here that we can use or example 0.55 as ratio to decrease WAF by 2.2 (increases RAF by 1.8)

### 4.3 Changing compactionMinFiles

Default value of compactionMinFiles is 3. Having following charts, we can make a conclusion, that we probably shouldn't change change this configuration value. Probably if we will extremly need to reduce WAF we can set this configuration to 7. It reduces WAF by 1.0, but increases RAF by 1.5.

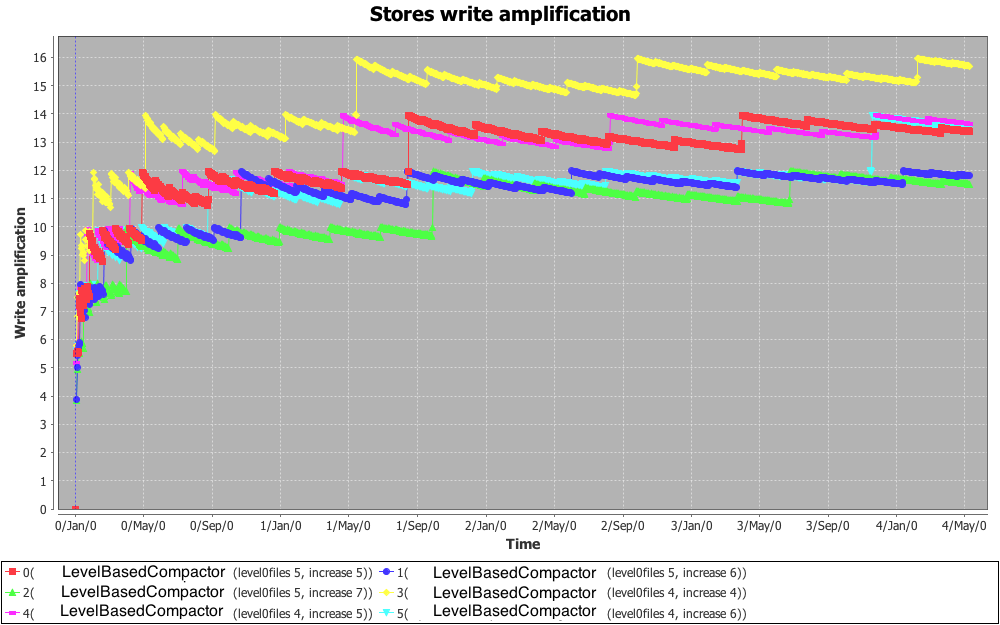
 [[Image:HBase--Compaction\_Simulator--ODS--finalminFilesSWAsmall.png]]

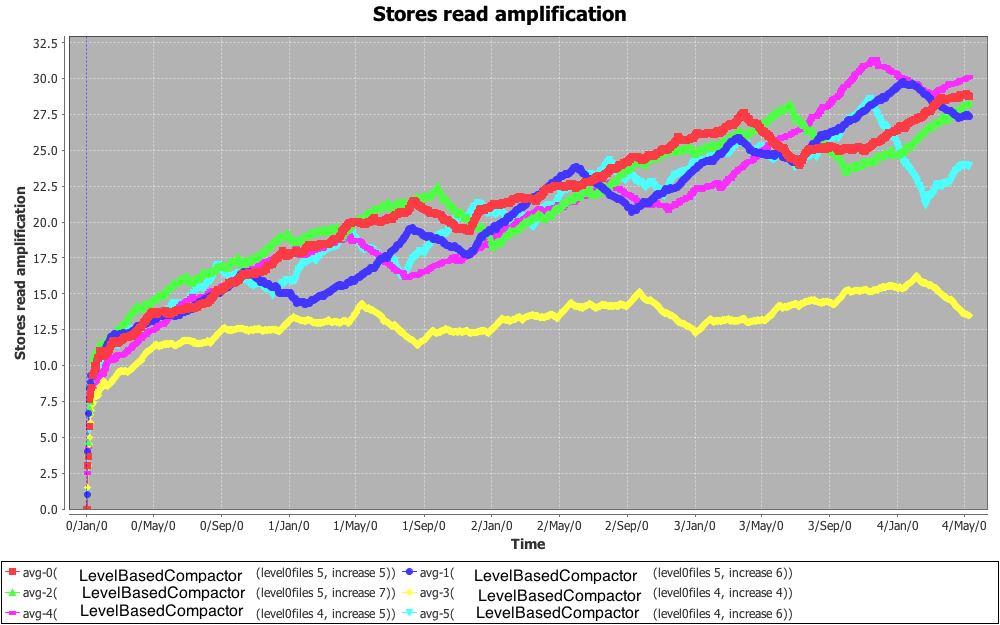
 [[Image:HBase--Compaction\_Simulator--ODS--finalminFilesSRAsmall.png]]

## 5.0 Testing different compaction algorithms

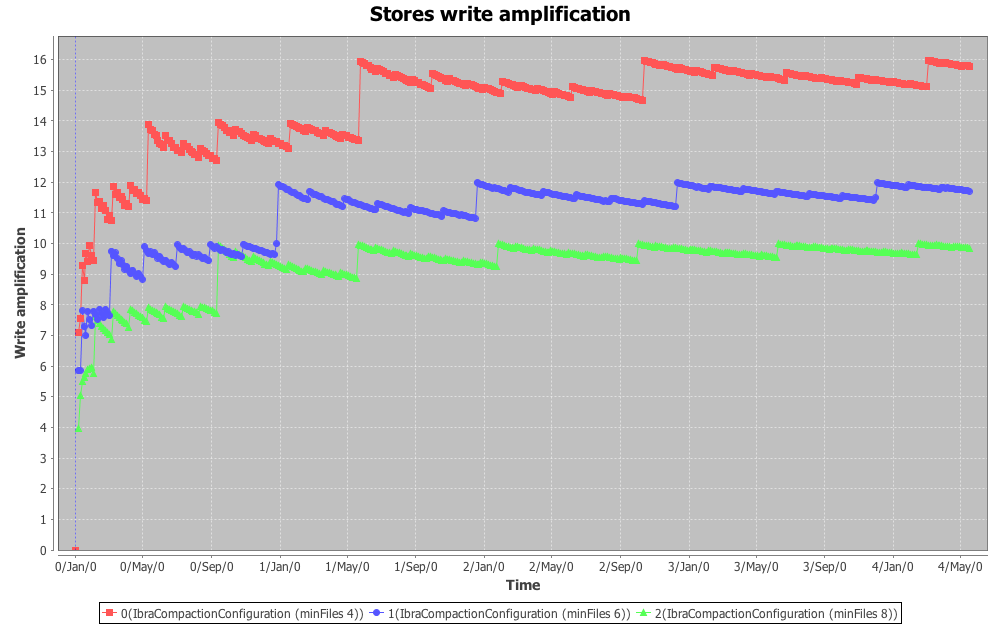
Let's compare different compaction algorithms with different configurations:

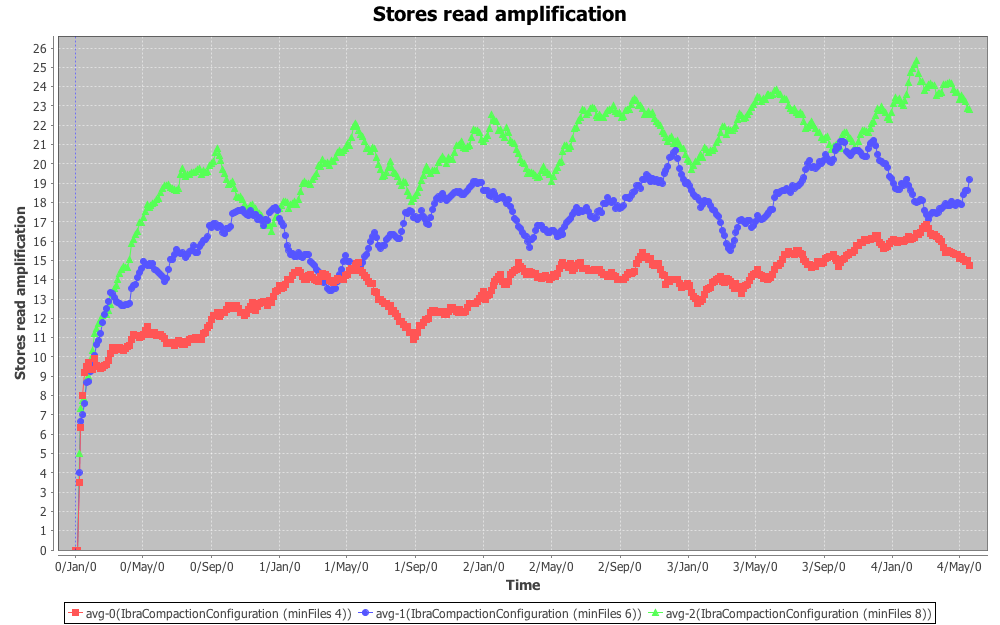
Here you can see level-based approach:

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--leveldbsSWA.png]]

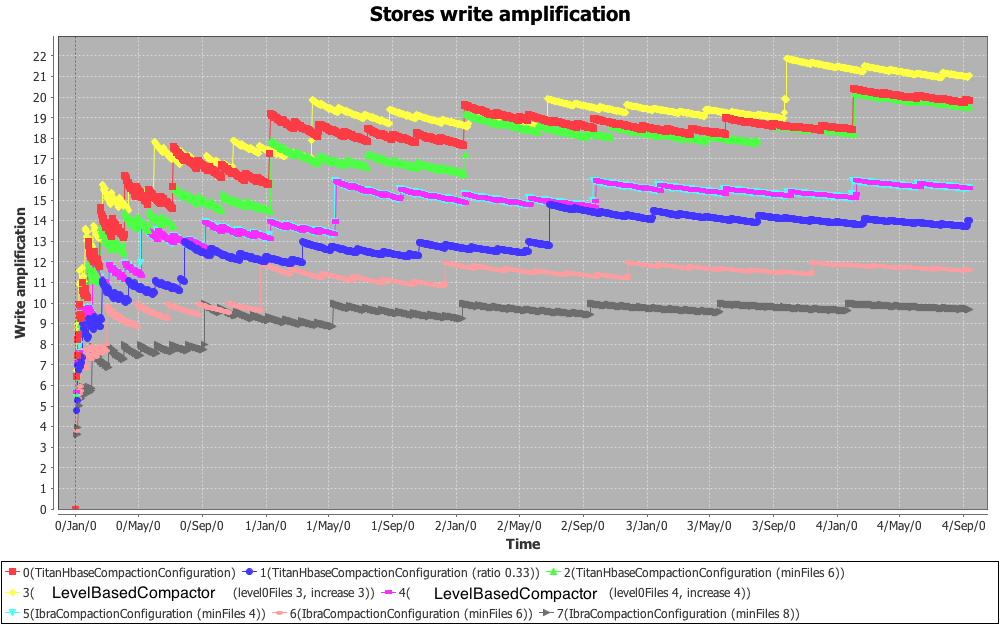
[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--leveldbsSRA.png]]

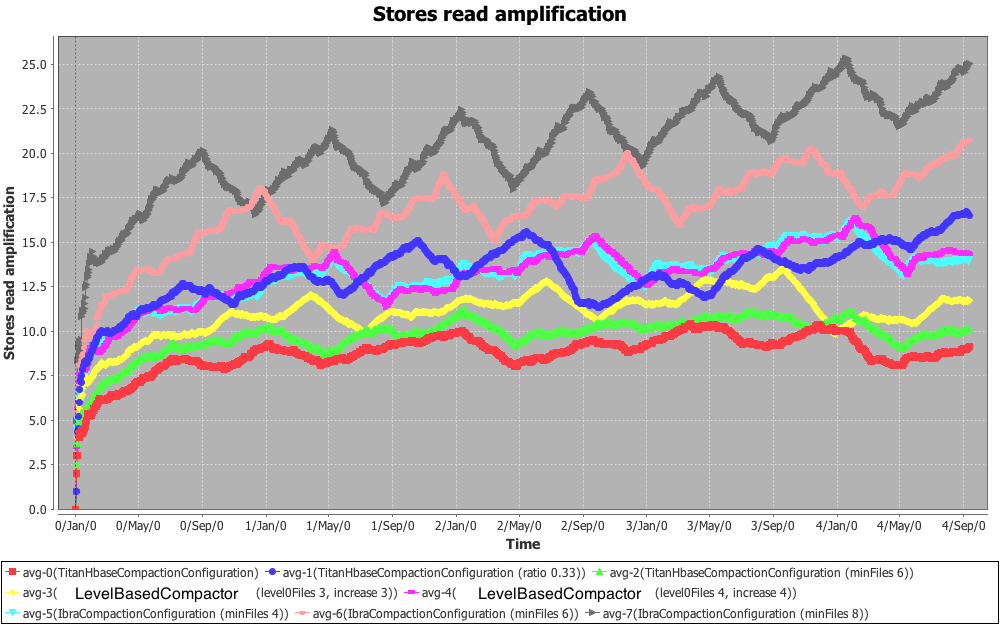
So let's take a closer look at IbraCompactor:

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--ibraminFilesSWA.png]]

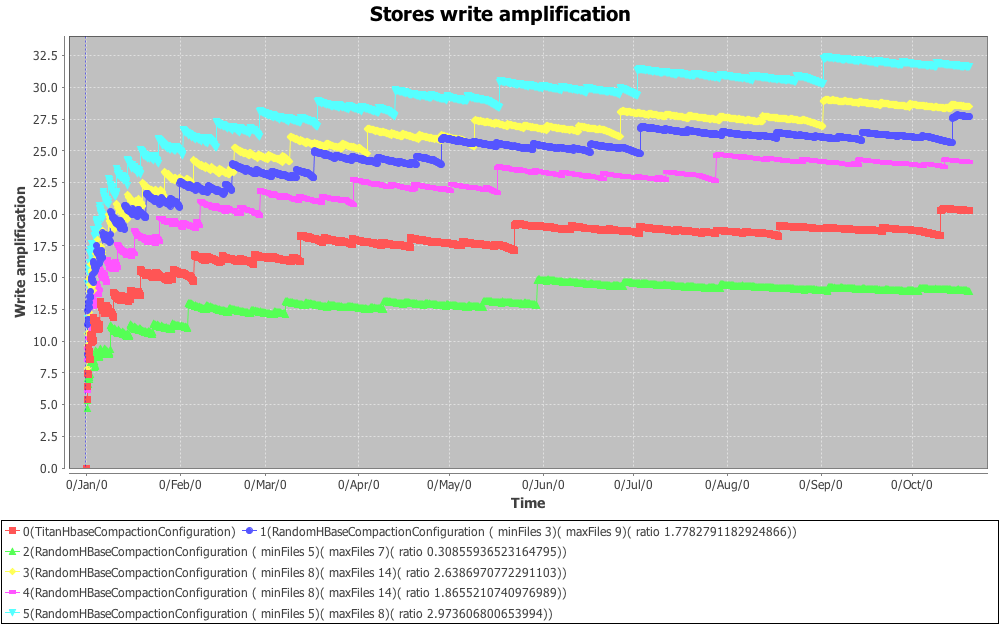
[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--ibraminFilesSRA.png]]

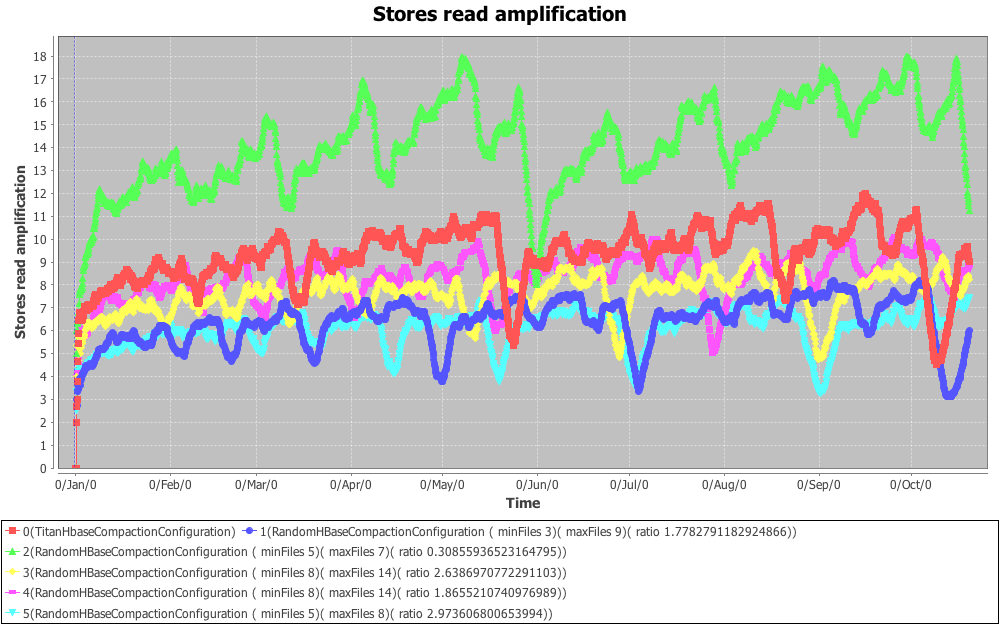
After simulating many different algorithms and configurations I came up with this chart, showing what can we have here. We can use TitanHBaseCompactor (ratio 0.33) [-6 SWA ,+5 SRA] or IbraCompactor (minFiles 6) [-8 SWA, +7.5 SRA] or IbraCompactor (minFiles 4) [-4 SWA, +4 SRA] - depending on how much do we need to decrease WAF and how much we can afford RAF increasing

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--mainSWAsmall.png]]

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--mainSRAsmall.png]]

Simulator also provides a way to have some random configurations for HBase native compaction algorithm:

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--SWA1.png]]

[[Image:HBase--Compaction\_Simulator--Different\_algorithms\_comparison--SRA1.png]]

## 6.0 Configurations analysis

There are 3 main configurations, that change the way how compactions works: *compaction ratio*, *compactionMinFiles* and *compactionMaxFiles*. *CompactionMinFiles* and *compactionMaxFiles* just validate compaction algorithm, setting restriction on number of files during compaction. And *compaction ratio* affects on compaction algorithm itself, because it is used to decide if compaction is needed.

First of all, let’s define some variables:

1. =

### 6.1 Compaction ratio analysis

Let’s start with *compaction ratio* analysis and investigation of how it affects WAF. To see how WAF depends on this configuration value, we need to ignore *compactionMinFiles* and *compactionMaxFiles*, setting *compactionMinFiles* = 2and *compactionMaxFiles = infinity.*

#### 6.1.1 Ratio is 1.0

So let's say that *compaction ratio* is 1.0. In the very beginning we have no files and after first flush we will have 1 file:

[1]

As far as we check if we need to run compaction algorithm after each flush, we do check now and see that we don't need to run compaction yet. We will keep flushing until we need to do a compaction:

[1] [1] ... [1] => [N], when ratio is 1.0, N is 2

The following table shows how compaction would be done:

|  |  |  |  |
| --- | --- | --- | --- |
| Flush id | Before flush | After flush | After compaction |
| 1 |  | [1] |  |
| 2 | [1] | [1][1] | [2] |
| 3 | [2] | [2][1] |  |
| 4 | [2][1] | [2][1][1] | [4] |
| 5 | [4] | [4][1] |  |
| 6 | [4][1] | [4][1][1] | [4][2] |
| 7 | [4][2] | [4][2][1] |  |
| 8 | [4][2][1] | [4][2][1][1] | [8] |
| 9 | [8] | [8][1] |  |
| 10 | [8][1] | [8][1][1] | [8][2] |
| 11 | [8][2] | [8][2][1] |  |
| 12 | [8][2][1] | [8][2][1][1] | [8][4] |
| 13 | [8][4] | [8][4][1] |  |
| 14 | [8][4][1] | [8][4][1][1] | [8][4][2] |
| 15 | [8][4][2] | [8][4][2][1] |  |
| 16 | [8][4][2][1] | [8][4][2][1][1] | [16] |
| 17 | [16] | [16][1] |  |
| 18 | [16][1] | [16][1][1] | [16][2] |

* compacting files are marked with red color

So as you can see major compactions are compactions with flush ids as powers of 2. Moreover, during compaction after i-th flush, we read/write

bytes, where

– is the Kronecker delta

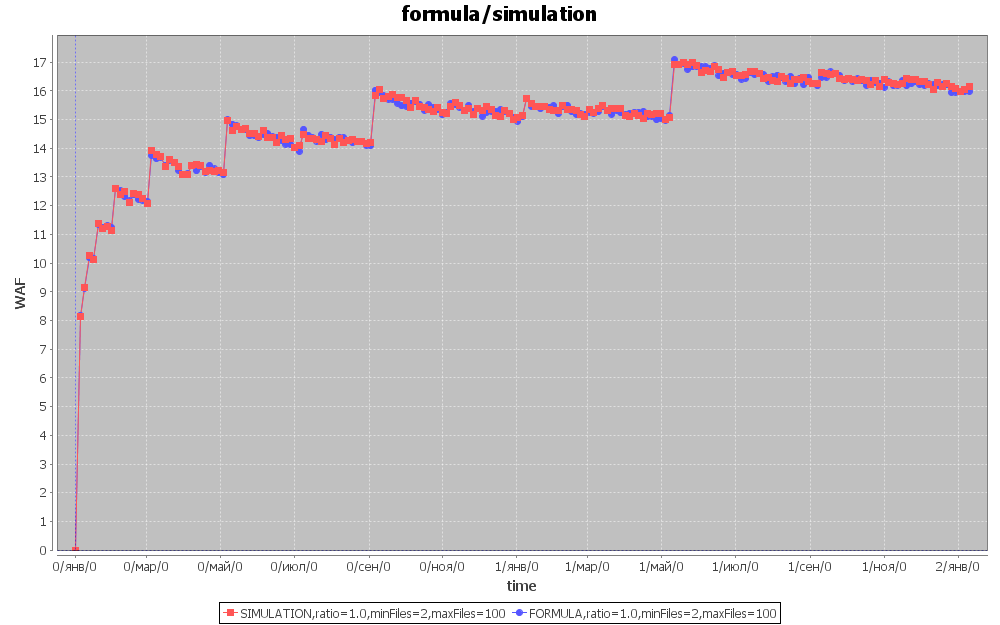
- is highest power of 2 dividing i, also known as <http://oeis.org/A006519>

- size of flushed file.

So, WAF after flushes will be (assuming there is no deletes due to TTLs or delete requests):

WAF() = ,

Using this formula, we get the following graph, and it is very similar to what HBase simulator shows:

 [[Image:HBase--Formula\_Simulation--comparison.png]]

#### 6.1.2 Ratio is 0.5

So let's say that *compaction ratio* is 0.5. In the very beginning we have no files and after first flush we will have 1 file:

[1]

As far as we check if we need to run compaction algorithm after each flush, we do check now and see that we don't need to run compaction yet. We will keep flushing until we need to do a compaction: Then there will be one more flush:

[1] [1] ... [1] => [N], when ratio is 1.0, N is 3

The following table show how compaction would be done:

|  |  |  |  |
| --- | --- | --- | --- |
| Flush id | Before flush | After flush | After compaction |
| 1 |  | [1] |  |
| 2 | [1] | [1][1] |  |
| 3 | [1][1] | [1][1][1] | [3] |
| 4 | [3] | [3][1] |  |
| 5 | [3][1] | [3][1][1] |  |
| 6 | [3][1][1] | [3][1][1][1] | [3][3] |
| 7 | [3][3] | [3][3][1] |  |
| 8 | [3][3][1] | [3][3][1][1] |  |
| 9 | [3][3][1][1] | [3][3][1][1][1] | [9] |
| 10 | [9] | [9][1] |  |
| 11 | [9][1] | [9][1][1] |  |
| 12 | [9][1][1] | [9][1][1][1] | [9][3] |
| 13 | [9][3] | [9][3][1] |  |
| 14 | [9][3][1] | [9][3][1][1] |  |
| 15 | [9][3][1][1] | [9][3][1][1][1] | [9][3][3] |
| 16 | [9][3][3] | [9][3][3][1] |  |
| 17 | [9][3][3][1] | [9][3][3][1][1] |  |
| 18 | [9][3][3][1][1] | [9][3][3][1][1][1] | [9][9] |

* compacting files are marked with red color

So as you can see major compactions are compaction with flush ids as powers of 3. Moreover, during compaction aftrt i-th flush, we read/write

bytes, where

– is the Kronecker delta

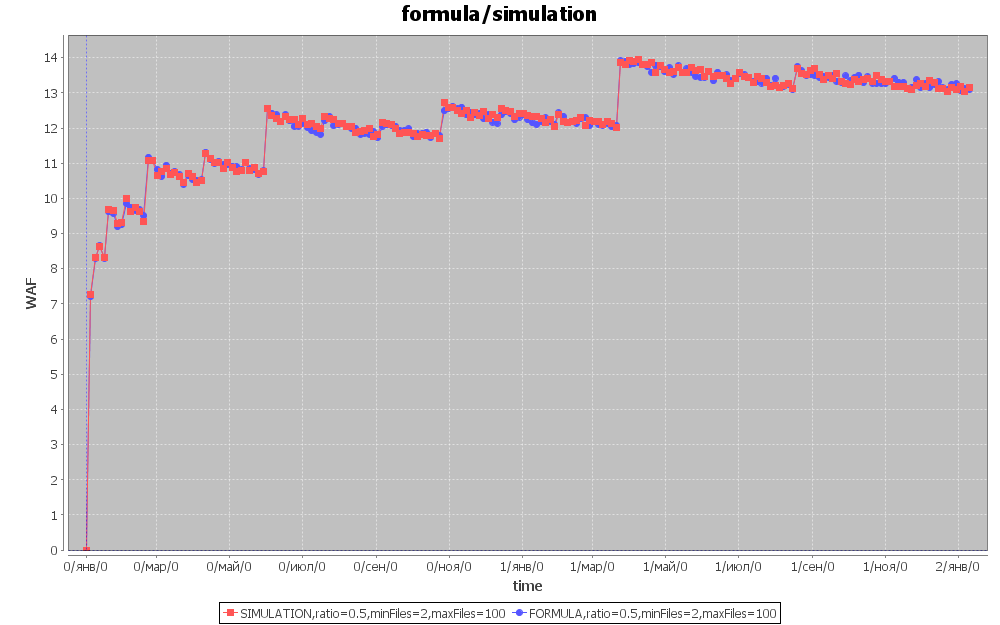
- is highest power of 3 dividing i, also known as <http://oeis.org/A038500>

- size of flushed file.

So, WAF after flushes will be (assuming there is no deletes due to TTLs or delete requests):

WAF() = ,

Using this formula, we get the following chart, and it is very similar to what HBase simulator shows:

 [[Image:HBase--Formula\_Simulation--comparison.png]]

#### 6.1.3 Any ratio

So, having all this in mind, we can ask ourselves, why when we use ratio 1.0 we use 2-based numeral system, and when ratio is 0.5, we use 3-based numeral system. The answer is inside compaction algorithm - let's say we have some files:

[a] [b] [c] [d] ...

We will take these files to compact if size(a) < size(b+c+d+...)\*ratio. Lets see the case when size(a) = size(b+c+d+...)\*ratio (because at first left part is bigger, but with flushing, at some moment both sides become equal, or left side becomes less)

So, size(a) = size(b+c+d+...)\*ratio

We take files [a] [b] [c] [d] .. and compact them into [COMP], and the size of this file is:

, when ratio is 1.0

, when ratio is 0.5

That is from where we get these numbers. So now having ratio we can somehow predict WAF:

WAF() = , where

- is highest power of dividing i

But there is one problem with this formula: when is integer, calculating is simple, but when is not integer, it becomes impossible to use this formula. For example when ratio is 0.4, = 3.5. Let’s see how compaction would be done this case:

|  |  |  |  |
| --- | --- | --- | --- |
| Flush id | Before flush | After flush | After compaction |
| 1 |  | [1] |  |
| 2 | [1] | [1][1] |  |
| 3 | [1][1] | [1][1][1] |  |
| 4 | [1][1][1] | [1][1][1][1] | [4] |
| 5 | [4] | [4][1] |  |
| 6 | [4][1] | [4] [1][1] |  |
| 7 | [4][1][1] | [4][1][1][1] |  |
| 8 | [4][1][1][1] | [4][1][1][1][1] | [4][4] |
| 9 | [4][4] | [4][4][1] |  |
| 10 | [4][4][1] | [4][4][1][1] |  |
| 11 | [4][4][1][1] | [4][4][1][1][1] |  |
| 12 | [4][4][1][1][1] | [4][4][1][1][1][1] | [4][4][4] |
| 13 | [4][4][4] | [4][4][4][1] |  |
| 14 | [4][4][4][1] | [4][4][4][1][1] | [14] |
| 15 | [14] | [14][1] |  |
| 16 | [14][1] | [14][1][1] |  |
| 17 | [14][1][1] | [14][1][1][1] |  |
| 18 | [14][1][1][1] | [14][1][1][1][1] | [14][4] |

* compacting files are marked with red color

To see how it works let’s look a little bit closer at and how it is calculated. To get, we should present number *i* in -based numeral system, and then is equal to radix of the smallest not-null position of this presentation. So now we can re-define:

- radix of the smallest not-null position of -based numeral system presentation of number *i*.

For example, if ratio is 1.0, is 2 and then

Let’s define radix of -based numeral system in i-th position as

Radixes of 2-based numeral system are:

|  |  |
| --- | --- |
| Position |  |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| … |  |
| i |  |

, smallest not-null (0-indexed) position is 2, and its radix is 4.

So all we need now is to present number *i* in -based numeral system. But there is no such a thing as numeral systems with non-integer bases. Here we can use mixed-radix numeral systems, exactly how it works in compaction algorithm. We will round up , using the following formula:

It is exactly how it is used in compaction algorithm – we do not perform compaction while

[size first file] is bigger than [size of other files]\*ratio

For example,

Compaction ratio = 0.4,

= ,

[size of first file] = sz1 = 1,

[size of other files] = sz2 = 2,

it is not enough to do a compaction, because 1 is bigger than 2\*0.4. And as far as we can’t perform a half-flush (sz1 to be equal to sz2\*ratio), we need to perform one more flush to be eligible to perform a compaction and size of sz2 will be equal to 3, and sz1+sz2 = .

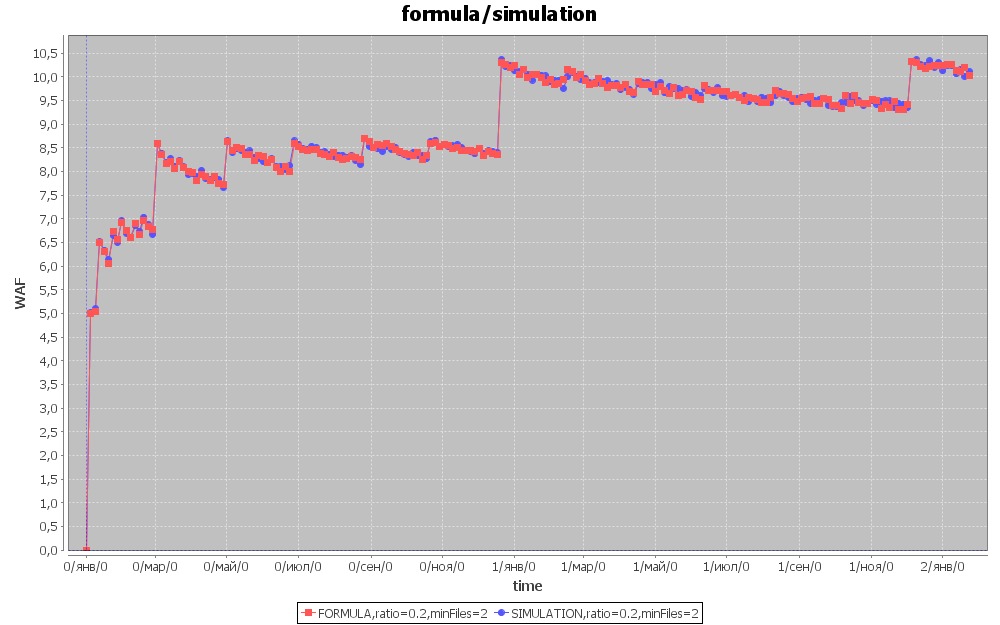
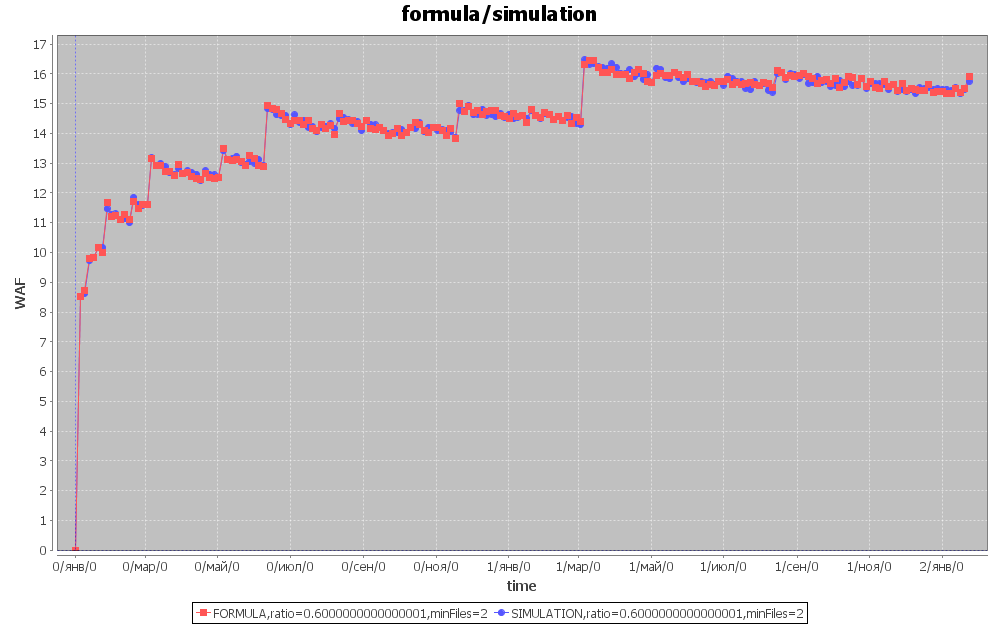
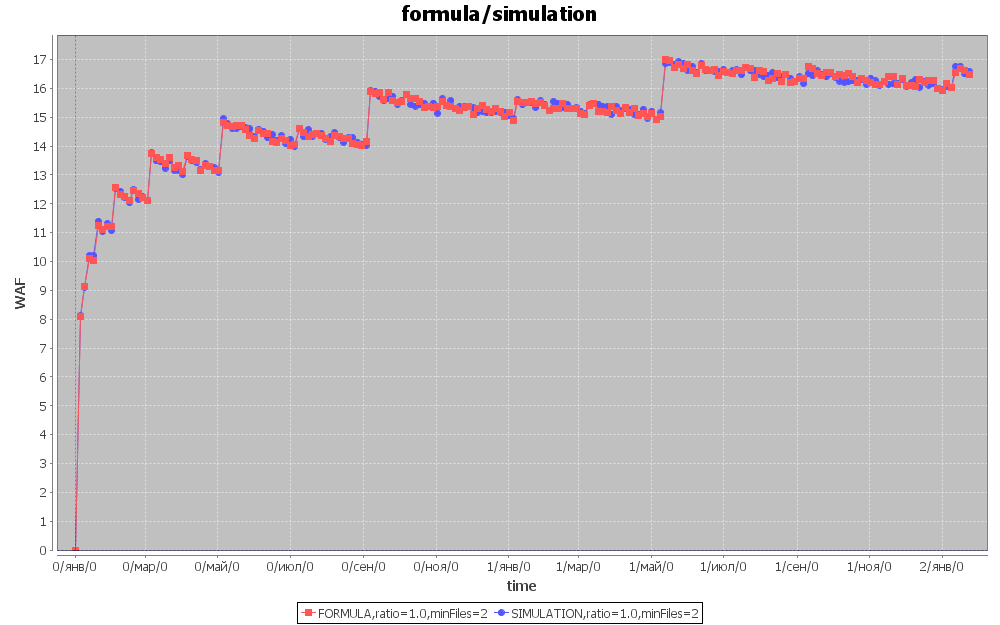
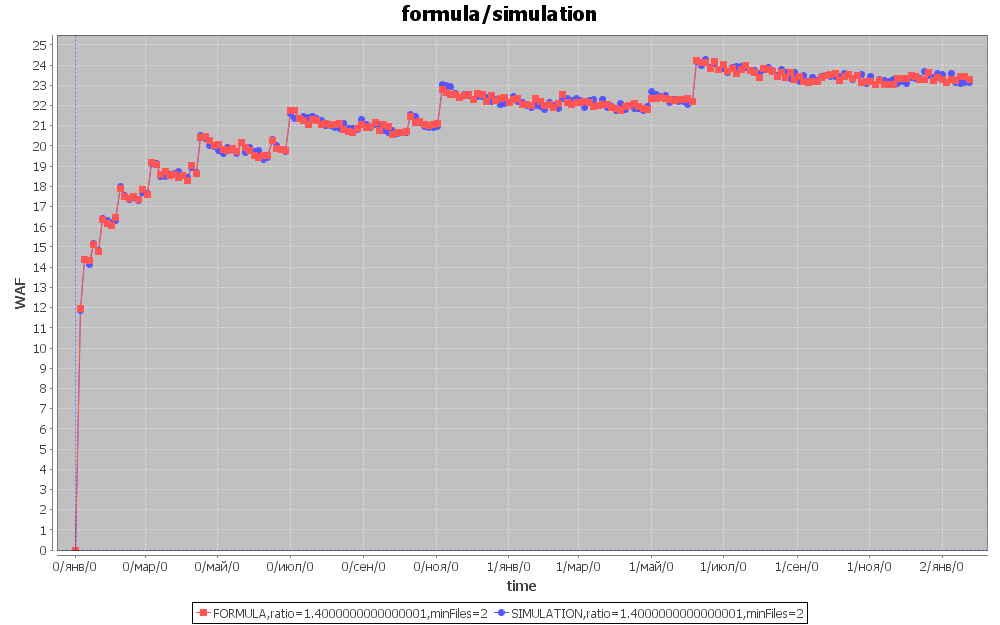
Here is the way how radixes are calculated:

|  |  |
| --- | --- |
| Position |  |
| 0 | 1 |
| 1 |  |
| … |  |
| i |  |

So if compaction ratio is 0.4 and is 3.5, radixes of -based numeral system are:

|  |  |
| --- | --- |
| Position |  |
| 0 | 1 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

We can check this formula with HBase compaction simulator. Following charts show difference between this formula and HBase compaction simulation. Here are presented charts with ratio set to 1.4, 1.0, 0.6, 0.2:



Red chart is generated with formula and blue is the result of simulation. As you can see it is the same.

### 6.2 CompactionMinFiles analysis

So now, having formula of WAF, analyzed with compaction ratio, now we can add compactionMinFiles to it.

#### 6.2.1 CompactionMinFiles <=

When we set compactionMinFiles to less or equal to , it is not used at all, and here is mathematical proof of this fact:

We want to prove that compaction algorithm always compacts more or equal to files. We will show that it is impossible to compact less than files:

1. We have a set of files:

And

1. To perform a compaction, we need the following condition to be true:

, so we need to prove that:

1. Having this condition:

we can rewrite is as:

It means that:

And as far as we need to prove that

We can rewrite it as follows:

Proved.

#### 6.2.2 CompactionMinFiles >

When we set compactionMinFiles value more than , in the very beginning, when we have just [1]-files, we can’t perform compaction because of compactionMinFiles restriction. So we just flush files until we have exactly compactionMinFiles [1]-files and then compact them:

[1][1]…[1] => [MN]

Then we will keep flushing and at some moment second compaction will happen:

[MN][1][1]…[1] => [MN][ MN]

It will keep happening until we have [MN]-files and MN [1]-files. Then we will compact it all:

[MN][ MN]…[ MN][1][1]…[1] => [ MN]

…

So as you can see all the difference that we get having compactionMinFiles configuration is the way how radixes are calculated:

|  |  |
| --- | --- |
| Position |  |
| 0 | 1 |
| 1 | MN |
|  |  |
|  |  |
| … | … |
| i |  |

Unfortunately, it is not all restrictions that came from this configuration value. Sometimes compaction algorithm needs to compact less than N files not in the very beginning. For example if ratio is 0.2, = 6, and compactionMinFiles is 7:

|  |  |  |  |
| --- | --- | --- | --- |
| Flush id | Before flush | After flush | After compaction |
| 1 |  | [1] |  |
| 2 | [1] | [1][1] |  |
| 3 | [1][1] | [1][1][1] |  |
| 4 | [1][1][1] | [1][1][1][1] |  |
| 5 | [1][1][1][1] | [4][1] |  |
| 6 | [1][1][1][1][1] | [1][1][1][1][1][1] |  |
| 7 | [1][1][1][1][1][1] | [1][1][1][1] [1][1][1] | [7] |
| 8 | [7] | [7][1] |  |
| … | … | … | … |
| 136 | [51][51][19][7][7] | [51][51][19][7][7][1] |  |
| 137 | [51][51][19][7][7][1] | [51][51][19][7][7][1][1] | [137] |
| … | … | … | … |

Compaction algorithm wants to do a compaction after 136-th flush, but it is impossible because it is only 6 files. Actually, there is a way to add this trick to formula, but it will make it too complex. On the other hand, this trick doesn’t make a big difference – it changes WAF a little bit and there is no need to consider it.

### 6.3 CompactionMaxFiles analysis

So now, having formula of WAF, analyzed with compaction ratio and compactionMinFiles, now we can add compactionMaxFiles to it. Ok, let’s see how it affects WAF. When we have some set of files to compact and size of this set is bigger than compactionMaxFiles value, we can’t compact. So algorithm takes some number (less or equal to compactionMaxFiles) of smallest files and compacts it. After that we do some flushes. And then again compact some smallest files and then again do flushes. And again and again until we do a major compaction:

[f11][f12]…[f1R][f21][f22]…[f2R]……[fn1][fn2]…[fnMN]

And after every compaction, we have several sets of equal files (and sizes of these sets are equal to R-1). So here is how it is happening:

We compact some files (and number of these files is biggest smaller or equal to compactionMaxFiles, let’s call this number T)

We flush files with size = 1

We compact some files (and number of these files is T)

We flush files with size = 1

…

In the end, we have less than T files. If number of files is less than compactionMinFiles, we flush 1-files until we have at least compactionMinFiles – and then compact it all. So it changes .

Using compactionMaxFiles comes out to following changes to :

Here is some proves and explanations about , and

1. *start* is first id when we need to add to . To explain how *start* is calculated, let’s define function as:

And is number of files that we use during major compaction to create file with size . So if we want to find smallest i that , then:

1. is also pretty simple to calculate – as it is mentioned above, after compacting in the end we have less than T files. If number of files is less than compactionMinFiles, we flush 1-files until we have at least compactionMinFiles – and then compact it all. So
2. When we compact files with corresponding and , we compact

* [1]-files or just 1 [1] file if
* Set of equal files, and size of this set is
* Sets of equal files, and sizes of these sets are

And as soon as we have MX restriction and is some integer value:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

And after this (and every) compaction we have set of files with equal sizes, then we again flush [1] files, and then this set and [1] files goes to next files during compaction, so set of [1] files goes to every k+1 files, so:

### 6.4 Gathering everything together

So, after all, we have formula for WAF:

WAF() , where

- flush id

- radix of the smallest not-null position of numeral system presentation of number *i*.

numeral system:

This formula was tested with HBase Compactions Simulator and it is 100% accurate to almost all the configurations. For others accuracy is , and average accuracy is 98%.

## 7.0 Editing Simulator

Simulator's code is formatted with default HBase eclipse code formatter

It is described here how to:

1) Add additional general configuration

2) Add additional compaction algorithm

3) Add additional compaction configuration

### 7.1 Adding additional general configuration

Code in Simulator is organized the way, so to add additional general configuration all you need to do is to go to Configuration class and then:

* add private field of this new configuration value
* add public getter and setter for it
* add adding this configuration to getFields and setFields map (in methods fillGetFields() and fillSetFields())

That's it. Now you can use this configuration in code, this configuration is already added to GUI, and can be changed by user

### 7.2 Adding additional compaction algorithm

To create new compaction algorithm you need to:

* create new class 'NewAlgorithm'Compactor to Model/Compactors/. This class should be inherited from AbstractCompactor
* create 'NewAlgorithm'CompactionConfiguration class for this 'NewAlgorithm'Compactor (see next paragraph about adding additional compaction configuration). Or you can use one of existing configurations if one of them fits you.
* add 'NewAlgorithm'CompactionConfiguration field to this class
* create constructor (see other other Compactors to see what should it be)
* override method StoreFileCollection selectFilesToCompact(StoreFileCollection storeFiles)

### 7.3 Adding additional compaction configuration

If you want just to have saved special configuration for some compaction algorithm that already exists, you can just create new 'NewAlgorithm'CompactionConfiguration, inherit it from base compaction configuration of compaction algorithm you want to add configuration of. Then set configuration you want in it's constructor. (See TitanHBaseCompactionConfiguration for clarifications)

If you are adding new compaction algorithm and now you want to add configuration for it, you need to:

* create class 'NewAlgorithm'CompactionConfiguration to Model/Compactors/CompactionConfigurations/
* inherit it from AbstractCompactionConfiguration
* add configuration values, you want to have here
* add getters and setters for these values
* override methods fillGetFields and fillSetFields (see other CompactionConfiguration classes for clarification)
* add declaration of this algorithm configuration to enum Model/Compactors/CompactionAlgorithm