

HFile format version 2 design document

Draft 0.1, 05/05/2011 (Mikhail Bautin, Liyin Tang)

Motivation.....	1
HFile format version 1 overview.....	2
Block index format in version 1.....	2
HBase file format with inline blocks (version 2).....	3
Overview.....	3
Unified version 2 block format.....	3
Block index in version 2.....	4
Root block index format in version 2.....	5
Non-root block index format in version 2.....	5
Bloom filters in version 2.....	5
File Info format in versions 1 and 2.....	6
Fixed file trailer format differences between versions 1 and 2.....	6

Motivation

We found it necessary to revise the HFile format after encountering high memory usage and slow startup times caused by large Bloom filters and block indexes in the region server. Bloom filters can get as large as 100 MB per HFile, which adds up to 2 GB when aggregated over 20 regions. Block indexes can grow as large as 6 GB in aggregate size over the same set of regions. A region is not considered opened until all of its block index data is loaded. Large Bloom filters produce a different performance problem: the first get request that requires a Bloom filter lookup will incur the latency of loading the entire Bloom filter bit array.

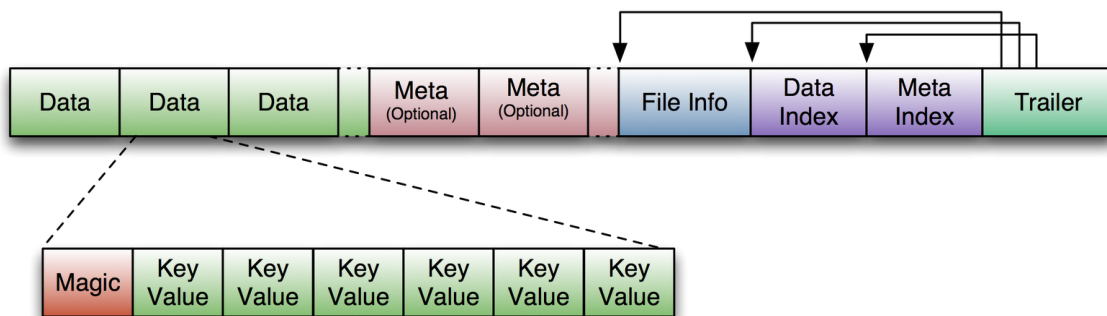
To speed up region server startup we break Bloom filters and block indexes into multiple blocks and write those blocks out as they fill up, which also reduces the HFile writer's memory footprint. In the Bloom filter case, filling up a block this means accumulating enough keys to efficiently utilize a fixed-size bit array, and in the block index case we accumulate an "index block" of the desired size. Bloom filter blocks and index blocks (we call these "inline blocks") become interspersed with data blocks, and as a side effect we can no longer rely on the difference between block offsets to determine data block length, as it was done in version 1.

HFile is a low-level file format by design, and it should not deal with application-specific details such as Bloom filters, which are handled at StoreFile level. Therefore, we call Bloom filter blocks in an HFile "inline" blocks. We also supply HFile with an interface to write those inline blocks.

Another format modification aimed at reducing the region server startup time is to use a contiguous “load-on-open” section that has to be loaded in memory at the time an HFile is being opened. Currently, as an HFile opens, there are separate seek operations to read the trailer, data/meta indexes, and file info. To read the Bloom filter, there are two more seek operations for its “data” and “meta” portions. In version 2, we seek once to read the trailer and seek again to read everything else we need to open the file from a contiguous block.

HFile format version 1 overview

As we will be discussing the changes we are making to the HFile format, it is useful to give a short overview of the current format. The existing HFile/StoreFile format is as follows:



(Image courtesy of Lars George, <http://www.larsgeorge.com/2009/10/hbase-architecture-101-storage.html>.)

Block index format in version 1

The block index in version 1 is very straightforward. For each entry, it contains:

- Offset (long)
- Uncompressed size (int)
- Key (a serialized byte array)
 - Key length as a variable-length integer (VInt)
 - Key bytes

The number of entries in the block index is stored in the fixed file trailer, and has to be passed in to the method that reads the block index. One of the limitations of the block index in version 1 is that it does not provide the compressed size of a block, which turns out to be necessary for decompression. Therefore, the HFile reader has to infer this compressed size from the offset difference between blocks. We fix this limitation in version 2, where we store on-disk block size instead of uncompressed size, and get uncompressed size from the block header.

HBase file format with inline blocks (version 2)

Overview

The version of HBase introducing the above features reads both version 1 and 2 HFiles, but only writes version 2 HFiles. A version 2 HFile is structured as follows:

File section using the Unified version 2 block format:			
“Scanned block” section	Data Block		
	...		
	Leaf index block / Bloom block		
	...		
	Data Block		
	...		
	Leaf index block / Bloom block		
	...		
	Data Block		
“Non-scanned block” section	Meta block	...	Meta block
	Intermediate Level Data Index Blocks (optional)		
File section not using the Unified version 2 block format:			
“Load-on-open” section	Root Data Index		
	Meta Index		
	<u>File Info</u>		
	<u>Bloom filter metadata (interpreted by StoreFile)</u>		
	Trailer (not including version)		Version

Unified version 2 block format

In the version 2 every block in the data section contains the following fields:

- 8 bytes: Block type, a sequence of bytes equivalent to version 1's "magic records". Supported block types are:

- DATA
- META
- BLOOM_CHUNK
- LEAF_INDEX
- INTERMEDIATE_INDEX
- Compressed size of the block's data, not including the header (int)
 - Can be used for skipping the current data block when scanning HFile data.
- Uncompressed size of the block's data, not including the header (int)
 - This is equal to the compressed size if the compression algorithm is NONE.
- File offset of the previous block of the same type (long)
 - Can be used for seeking to the previous data/index block.
- Compressed data (or uncompressed data if the compression algorithm is NONE).

The above format of blocks is used in the following HFile sections:

- Scanned block section. The section is named so because it contains all data blocks that need to be read when an HFile is scanned sequentially. Also contains leaf block index and Bloom chunk blocks.
- Non-scanned block section. This section still contains unified-format v2 blocks but it does not have to be read when doing a sequential scan. This section contains meta blocks and intermediate-level index blocks.

We are supporting meta blocks in version 2 the same way they were supported in version 1, even though we do not store Bloom filter data in the meta index anymore.

Block index in version 2

There are three types of block indexes in HFile version 2, stored in two different formats (**root** and **non-root**):

- Data index — version 2 multi-level block index, consisting of:
 - Version 2 **root** index, stored in the data block index section of the file
 - Optionally, version 2 intermediate levels, stored in the **non-root** format in the data index section of the file.
 - Intermediate levels can only be present if leaf level blocks are present.
 - Optionally, version 2 leaf levels, stored in the **non-root** format inline with data blocks
- Meta index — version 2 **root** index format only, stored in the meta index section of the file
- Bloom index — version 2 **root** index format only, stored in the “load-on-open” section.

Root block index format in version 2

This format applies to:

- Root level of the version 2 data index;
- Entire meta and Bloom indexes in version 2, which are always single-level.

A version 2 root index block is a sequence of entries of the following format, similar to entries of a version 1 block index, but storing on-disk size instead of uncompressed size.

- Offset (long)
 - This offset may point to a data block or to a deeper-level index block.
- On-disk size (int)
- Key (a serialized byte array)
 - Key (VInt)
 - Key bytes

A single-level version 2 block index consists of just a single root index block. To read a root index block of version 2, one needs to know the number of entries. For the data index and the meta index the number of entries is stored in the trailer, and for the Bloom index it is stored in the compound Bloom filter metadata.

Non-root block index format in version 2

This format applies to intermediate-level and leaf index blocks of a version 2 multi-level data block index. Every non-root index block is structured as follows.

- *numEntries*: the number of entries (int).
- *entryOffsets*: the “secondary index” of offsets of entries in the block, to facilitate a quick binary search on the key (*numEntries* int values).
- Entries. Each entry contains:
 - Offset of the block referenced by this entry in the file (long)
 - On-disk size of the referenced block (int)
 - Key. The length can be calculated from *entryOffsets* and the total size of the non-root index block.

Bloom filters in version 2

In contrast with version 1, in a version 2 HFile Bloom filter metadata is stored in the load-on-open section of the HFile for quick startup. Bloom filter formats in an HFile of version 2 can be of the two types shown below. A compound Bloom filter is only used when enabled in the configuration and when a single Bloom filter would be too large. Similar fields between these two cases are highlighted with the same color, but data type changes are emphasized with bold.

- A single [ByteBloomFilter](#). In this case the Bloom filter metadata section in version 2 contains a concatenation of what BLOOM_FILTER_META and BLOOM_FILTER_DATA blocks would contain in a version 1 file:
 - Magic string
 - Bloom filter version = 1 (int)
 - Byte size of the Bloom filter (int)
 - Number of hash functions used (int)
 - Hash function type (int, 0 for JENKINS_HASH, 1 for MURMUR_HASH)
 - The number of keys inserted (int)
 - Bloom filter data (a byte array)
- A compound Bloom filter.
 - Magic string (BLMFBLK2)
 - Bloom filter version = 3 (int)
 - Number of chunks (int)
 - Total byte size of all chunks of this Bloom filter (long)
 - Number of hash functions used (int)
 - Hash function type (int)
 - Total number of keys inserted into all chunks (long)
 - Maximum number of keys per chunk (long)
 - Total maximum number of keys in all chunks (long)
 - Bloom block index in the [version 2 root block index format](#).

We chose to keep the

File Info format in versions 1 and 2

A serialized [HbaseMapWritable](#) (essentially a map from byte arrays to byte arrays) with the following keys:

hfile.LASTKEY	The last key of the file (byte array)
hfile.AVG_KEY_LEN	The average key length in the file (int)
hfile.AVG_VALUE_LEN	The average value length in the file (int)
hfile.COMPARATOR	Comparator class name (UTF-8 string)

File info format did not change in version 2. However, we moved the file info to the final section of the file, which can be loaded as one block at the time the HFile is being opened.

Fixed file trailer format differences between versions 1 and 2

The following table shows common and different fields between fixed file trailers in versions 1 and 2. Note that the size of the trailer is different depending on the

version, so it is "fixed" only within one version. However, the version is always stored as the last four-byte integer in the file.

Version 1	Version 2
File info offset (long)	
Data index offset (long)	loadOnOpenOffset (long) <i>The offset of the section that we need to load when opening the file.</i>
Number of data index entries (int)	
Meta index offset (long) <i>This field is not being used by the version 1 reader, so we removed it from version 2.</i>	
Number of meta index entries (int)	
Total uncompressed bytes (long)	
numEntries (int)	numEntries (long)
Compression codec: 0 = LZO, 1 = GZ, 2 = NONE (int)	
	The number of levels in the data block index (int)
	firstDataBlockOffset (long) <i>The offset of the first first data block. Used when scanning.</i>
	lastDataBlockEnd (long) <i>The offset of the first byte after the last key/value data block. We don't need to go beyond this offset when scanning.</i>
Version: 1 (int)	Version: 2 (int)