Cassandra Storage Format
Proposal for 3.1
Version 1

Benedict Elliott Smith

January 2015

Abstract

This document is a draft proposal for a generalised replacement for the sstable format used in Cassandra. The goal is to deliver a framework building on top of a number of abstracted components, with which any efficient sstable representation can be encapsulated. It is designed to directly align with modern CQL data modelling, however it should also support thrift models at least as efficiently.

1 Outline

This section will outline the components and some associated concepts that are common to

Definitions

PartitionEpoch, *RowEpoch*

Against each tier of our structure we will encode the minimum timestamp found in the next tier, so that each may be encoded more efficiently. The expectation is that this will permit many cells to store either only a few bytes, a single bit, or no data at all, to represent their timestamp.

ColId

A translation from column to a unique integer id

Components

These components will each have more than one implementation.

PartitionIndex

One per sstable. Performs a translation from $PartitionKey$ to a set of addresses in $DataFile$ that may contain data for the key. This translation is permitted to yield false positives, but should expect to produce either zero or one result on average, with no false negatives. Multiple keys may map to the same location in $DataFile$, which can be exploited by an implementation to produce a compact mapping. These keys will generally be adjacent, but some may be out-of-order due to page packing. Its persistence is managed separately.

$RowIndex$

One per partition. Accessed sequentially, located by a $PartitionIndex$ lookup, it performs a translation from $RowKey$ to a $RowEpoch$ and a position to provide to each $ValueStore_N$ to produce cell values. It is stored in $DataFile$.

$ValueStore$

One per partition. A sequential chunk of bytes that combines the position retrieved from the $RowIndex$ with an offset stored in $DataFile_0$, the queried columns, the $ColId$ translation and the $RowEpoch$ to return cell values. We assume a property Pos can be queried prior to insertion of a new row that returns the value we will store in $RowIndex$ for future lookups. It is stored in $DataFile$.

2 Implementation Details

2.1 Notation Key

$X \Leftarrow Y$	Write/Add/Append Y To X
$X \dashleftarrow Y$	Read X From Y
$X \leftarrow Y$	Set X to Y
$\langle A, B\rangle$	A tuple containing the values A and B
$\{Item \mid Constraint\}$	The set of all $Item$ produced by $Constraint$
S_N	$\equiv \{S_n \mid n \in N\}$
$Stmt_x \mid \forall x \in X$	Execute statement S for each x in X
$\lvert X\rvert$	$\equiv$ **card**(X), i.e. the number of elements in X

2.2 Core Outline

This section will outline the core algorithm using the component definitions provided in the previous section. A basic description of the algorithm follows.

Description

```
Write($Partitions$)
```

Algorithm 1: Writing