Bloom filters

To avoid checking every SSTable data file for the partition being requested, Cassandra employs a data structure known as a bloom filter.
Bloom filters are a probabilistic data structure that allows Cassandra to determine one of two possible states: - The data definitely does not exist in the given file, or - The data probably exists in the given file.

While bloom filters can not guarantee that the data exists in a given SSTable, bloom filters can be made more accurate by allowing them to consume more RAM. Operators have the opportunity to tune this behavior per table by adjusting the `bloom_filter_fp_chance` to a float between 0 and 1.

The default value for `bloom_filter_fp_chance` is 0.1 for tables using LeveledCompactionStrategy and 0.01 for all other cases.
Bloom filters are stored in RAM, but are stored offheap, so operators should not consider bloom filters when selecting the maximum heap size. As accuracy improves (as the `bloom_filter_fp_chance` gets closer to 0), memory usage increases non-linearly - the bloom filter for `bloom_filter_fp_chance = 0.01` will require about three times as much memory as the same table with `bloom_filter_fp_chance = 0.1`.

Typical values for `bloom_filter_fp_chance` are usually between 0.01 (1%) to 0.1 (10%) false-positive chance, where Cassandra may scan an SSTable for a row, only to find that it does not exist on the disk. The parameter should be tuned by use case:

- Users with more RAM and slower disks may benefit from setting the `bloom_filter_fp_chance` to a numerically lower number (such as 0.01) to avoid excess IO operations
- Users with less RAM, more dense nodes, or very fast disks may tolerate a higher `bloom_filter_fp_chance` in order to save RAM at the expense of excess IO operations
- In workloads that rarely read, or that only perform reads by scanning the entire data set (such as analytics workloads), setting the `bloom_filter_fp_chance` to a much higher number is acceptable.

The bloom filter false positive chance is visible in the `DESCRIBE TABLE` output as the field `bloom_filter_fp_chance`. Operators can change the value with an `ALTER TABLE` statement: :

```
ALTER TABLE keyspace.table WITH bloom_filter_fp_chance=0.01
```

Operators should be aware, however, that this change is not immediate: the bloom filter is calculated when the file is written, and persisted on disk as the Filter component of the SSTable. Upon issuing an `ALTER TABLE` statement, new files on disk will be written with the new `bloom_filter_fp_chance`, but existing sstables will not be modified until they are compacted - if an operator needs a change to `bloom_filter_fp_chance` to take effect, they can trigger an SSTable rewrite using `nodetool scrub` or `nodetool upgradesstables -a`, both of which will rebuild the sstables on disk, regenerating the bloom filters in the progress.