**HOW TO CHOOSE SS TABLES TO COMPACT :**

SSTables are separated by density leveling in levels determined by the fan factors of the compaction setup. The amount of SSTables on a level, however, cannot be used as a trigger because there may be non-overlapping SSTables, unlike in size leveling where SSTables are anticipated to cover the entire token area. In this case, read queries are less effective. Use sharding, which decreases the size of individual compaction operations and executes many compactions on a level concurrently, to address this. The amount of overlapping SSTables in a bucket dictates what has to be done with the non-overlapping area, which needs to be divided into several buckets. The chosen collection of SSTables that will be compressed together is called a bucket.

Make a minimal list of overlap sets first that satisfies the following criteria:   
1. Two non-overlapping SSTables are never placed in the same set.  
2.There is a set in the list that contains both SSTables if they overlap.   
SSTables are arranged in the list in successive positions.   
  
Another way to frame the second requirement is to say that there is a set in the list that contains all SSTables whose range spans any given point in the token range. Put differently, the overlap sets provide us with the maximum number of SSTables that must be consulted in order to read any key; this is the read amplification that our trigger t seeks to regulate. We don’t calculate or store the exact spans the overlapping sets cover, only the participating SSTables. The sets can be obtained in O(nlogn)

time.

For instance,

we calculate a list of overlap sets that are ABD and BCD if SSTables A, B, C, and D cover, respectively, the tokens 0-3, 2-7, 6-9, and 1-8. Since A and C don't overlap, they have to belong in different sets. As a result of their overlap at token 2, A, B, and D must appear in at least one set; the same is true for B, C, and D at token 7. The combination is previously present in the set ABD, although only A and D overlap at 1.   
If and only if a set's element count is at least as large as st, these overlap sets are adequate to determine whether or not a compaction should be done. But, it's possible that the compaction will require more SSTables than just this set.

Our sharding strategy might result in the creation of SSTables that span variously sized shards for the same level. The situation of leveling compaction is one glaring illustration. As SSTables enter at a certain density in this instance, they split in two near the middle of the token range because the final SSTable is twice as large as the initial density following the first compaction. We will have distinct overlap sets between the two older SSTables and the new one when it enters the same level as the other one. The next compaction that is triggered must choose both overlap sets in order to be efficient.

The overlap sets will transitively extend with all nearby ones that share some SSTable in order to handle scenarios of partial overlap. As a result, a chain of overlapping SSTables ties the created set of all SSTables to the original set. The compaction bucket is formed by this enlarged collection.Every SSTable is compacted in the compaction bucket during regular operations. We might impose a cap on the number of overlapping sources we compact if compaction proceeds very slowly. To ensure that if an SSTable is included in this compaction, all older ones are also included to maintain temporal order. In that scenario, we employ the collection of oldest SSTables that would select at most limit-many in any included overlap set.

HOW TO SELECT A COMPACTION RUN :

The number of SSTables that overlap on a given key defines the read amplification of queries, which is the target of compaction techniques. When compaction is delayed, the compaction bucket with the highest overlap among the options is chosen for optimal efficiency. In each stage, select a choice uniformly and at random if there are several options. The lowest level should be chosen since it is anticipated to cover a bigger portion of the token space for the same amount of work.

On some level, this approach stops the accumulation of SSTables under prolonged stress, which can occasionally occur with legacy solutions. Older tactics could have resulted in L0 using up all resources and SSTables building up on L1. With UCS, a tiered hierarchy based on the lowest overlap they can maintain for the load is maintained, and a steady state where compactions always consume more SSTables than the designated threshold and fan factor is achieved.

WHAT IS MAJOR COMPACTION :

According to the principles underlying UCS, a major compaction is an operation that combines all SSTables with (transitive) overlap and splits the result on shard boundaries suitable for the anticipated density that results.

Put otherwise, b concurrent compactions, each containing all SSTables covered in each of the foundation shards, will arise from a main compaction. The outcome will be divided along shard boundaries, the number of which is determined by the total volume of data in each shard.

SHARD COUNT :

**Synthetic Sharding Strategy: Shard Table**

**Pros**

* Always works.
* Easy to parallelize (can be writing to shards in parallel).
* Very very common and therefore battle tested.

**Cons**

* May have to do shards of shards for particularly large partitions.
* Hard for new users to understand.
* Hard to use in low latency use cases (but so are REALLY large partitions, so it’s a problem either way).

The bare minimum, or b, of shards, applied to the densest tiers. For the lowest levels, this provides the least compaction concurrency. Because every item of data must pass through L0, a low number would provide larger L0 sstables but may also restrict the total possible write throughput.

4 is the default (1 for system tables, or if more than one data location is specified).