The LSM-Trie KV Store focuses on getting write amplification down into single digits – compared to a 5 level LevelDB store with a write amplification of 55, a 5 level LSM-Trie can achieve a write amplification of 5! LSM-Trie has 5 main levels with levels 1-4 having 8 sublevels and level 5 having up to 80 sublevels. These 112 levels let it store up to 10TB of data. It is hash based, only using an index for a few large KV items. LSM-Trie utilizes a prefix tree structure that is constructed by putting the key through a SHA1 hash and utilizing pieces of the 160bit key to traverse the prefix tree and find the KV pair in an HTable. HTables are 32MB and consist of 4KB buckets that store KV pairs. The buckets are 4KB to match disk blocks so they can be read with a single disk read. 16bit bloom filters are utilized to achieve a 5% false positive rate over the 112 levels. When all the sublevels are filled at a level k, the sublevels are sorted and written as a sublevel of level k+1. In this way, pushing data to a lower level only costs 1 write instead of an exponential amount. Data can be retrieved by traversing the trie based on the hash, checking bloom filters with a single read as they are clustered together (bloomcluster), and then reading the single block that contains the KV item.

1. “In the meantime, for some KV stores, such as SILT [24], major efforts are made to optimize reads by minimizing metadata size, while write performance can be compromised without conducting multi-level incremental compactions” Explain how high write amplifications are produced in SILT.
   a. SILT utilizes a log that then gets sorted into a hashstore. The hashstore is then merged with the entire sortedstore on the disk. This results in the write amplification becoming huge as the ratio between the amount of data being written and the amount of data that has to be merged continues growing – the write amplification can get to tens of thousands.
2. “Note that LSM-trie uses hash functions to organize its data and accordingly does not support range search.” Do FAWN and LevelDB support range search?
   a. FAWN is hash based, so it cannot support range search
   b. LevelDB uses SSTables that hold sorted key-value pairs, indices of the SSTables are block ranges, so it supports range search.
3. Use Figure 1 to explain the difference between linear and exponential growth patterns.
   a. Linear growth is illustrated here as sublevels – when new data is added, it can be put in a sublevel and the growth cost is linear as the ratio of data being added is 1:1. Exponential growth is between levels illustrated here – growth into a lower level is an
exponential difference in cost as it is going from 1:10 in the case of LevelDB.

4. “Because 4KB block is a disk access unit, it is not necessary to maintain a larger index to determine byte offset of each item in a block.” Show how a lookup with a given key is carried out in LevelDB?
   a. Binary search on index, see if it is in memtable.
   b. Binary search each successive level for blocks that contain the range of the key.
   c. Check Bloom Filter to see if a SSTable might contain the key
   d. Retrieve the value from the SSTable if the key exists, otherwise continue from b.

5. “Instead, we first apply a cryptographic hash function, such as SHA-1, on the key, and then use the hashed key, or hashkey in short, to make the determination.” Assuming a user-provided key has 160 bits, what’s the issue if LSM-trie used the user keys, instead of hashed keys, in its data structure and operations?
   a. A user key could result in an unbalanced tree being formed, where in the worst case a linked list of all the KV items would be constructed. As mentioned in the paper: “It is known that using a cryptographic hash function allows each bucket to have statistically equal chance to receive a new item, and item count in each bucket follows a normal distribution.” A hash functions such as SHA1 provides a normal distribution that will create a mostly balanced tree.

6. “Among all compactions moving data from Lk to Lk+1, we must make sure their key ranges are not overlapped to keep any two SSTables at Level L k+1 from having overlapped key ranges. However, this cannot be achieved with the LevelDB data organization ...” Please explain why LevelDB cannot achieve it?
   a. As mentioned in the paper about LevelDB: “...the sorted KV-items at each sub-level are placed into the SSTables according to the tables’ fixed capacity (e.g., 32 MB). The key range size of an SSTable can be highly variable and the ranges’ distribution can be different in different sublevels. Therefore, ranges of the aforementioned compactions are likely to be overlapped.” Items that come in will be placed in a sorted SSTable until it gets full. As items get merged to lower levels, the ranges that each SSTable covers will change and will be different between levels.
7. Use Figures 2 and 3 to describe the LSM-trie’s structure and how compaction is performed in the trie.
   a. LSM-Trie is a prefix tree based on the hashed key. The prefix of the hash is used in increments as illustrated in Figure 2 to traverse the tree. From the root node, the first 3 numbers say which branch to take, the next 3 numbers say the next branch, etc. As this is based on the hash value, no index is necessary to traverse the tree and find the right bucket that contains the KV item.
   
   ![](image)
   
   **Figure 2:** A trie structure for organizing SSTables. Each node represents a table container, which contains a pile of SSTables.
   
   b. Compaction is performed in this way: the sublevels in level \( k \) are sorted based on their hash values, and the sorted result is then pushed to level \( k+1 \) as a new sublevel at the top of level \( k+1 \).
   
   ![](image)
   
   **Figure 3:** A compaction operation in the trie.