Optimized Min-heap based Similarity Detection for Delta Encoding

Described is a technique for detecting approximate similarity between different chunks of data which enables the use of Delta Encoding techniques to reduce data volume, in addition to Deduplication based on exact matches. While Similarity based Delta Encoding has been used before, the algorithm described here uses a unique variant of the MinHash technique to compute a hash-based Similarity Index value for a data chunk. This value can then be compared to the values of the other chunks to detect similar chunks.

The algorithm uses a variant of the common MinHash technique that is used by search engines, document content mining and plagiarism detection tools. In traditional MinHash the content is split into small semantic elements and each element is hashed. Then the numerically lowest set of K hash values are selected to create a K-min-values set or sketch. A single super-hash identifier can then be derived by hashing the hash values in this set. Two sections of data having the same super-hash contain the least K elements in common and are considered similar to each other.

While MinHash can conceptually be used to detect similar chunks for Delta Encoding during the Data Deduplication process, it needs certain modifications. Traditional MinHash is applicable to scenarios where the domain or type of the data is restricted and known. Examples include web HTML files, English or other language text, Images etc. Data Deduplication deals with arbitrary bytes in data chunks where the nature of the data may not be known beforehand or even be unpredictable. So we need to modify the minhashing approach to deal with raw data bytes. The working of this idea can be summarized in the following steps:

1. Consider the raw bytes in the chunk as a sequence of integers typically 8-byte, 64-bit wide. Using 4-byte, 32-bit integers is also fine and can be a bit more accurate but at the cost of performance.
2. Endianness does not matter here. The byte stream can be processed in the processor's native endian format for performance.
3. We then construct a min-heap out of this logical sequence of integers. A standard in-place heapify algorithm can be used for good performance.
4. K lowest integer values get collected into the min-heap during the heapify process. The value of 'K' is chosen based on the extent of similarity desired. For example if the chunk has 100 integers and the desired minimum similarity match is 40% then K equals 40 integers. So 40 lowest valued integers are copied into the min-heap from the total set of 100. We can call this a K-min-heap sketch.
5. Finally a hash is computed over the raw bytes of the min-heap. This hash is the K-similarity hash for the chunk. If another data chunk has the same hash then the two chunks are considered to be at least 40% similar to each other.
6. These chunks can then be passed to the Delta Encoding process for encoding one as the diff of the other.
7. The hash function can be any hashing algorithm with good distribution properties.

So the crux of this algorithm is to construct min-heaps over raw data and hash the heaps directly without any subsequent sorting. Note than unlike the traditional MinHash there is no splitting of data into semantic elements and subsequent element-hash and super-hash computation.

Empirically this algorithm has high-performance and provides excellent results with virtually no false positives. In the above 40% similarity example a false positive can occur if min-heap hashes of two

chunks match but the chunks are less than 40% similar. An extension of this technique can be applied in the cases if chunks are further compressed(using an LZ-style compression for eg.) after Deduplication and Delta Encoding. There are two approaches to the compression listed below:

1. Compress each chunk independently. No special considerations apply in this case.
2. Group several chunks together into a segment and compress the whole segment.
    - This approach results in better compression.
    - However doing similarity based Delta Encoding in this case can reduce subsequent compression ratio using a normal compression algorithm like LZW, Zlib, Bzip2 etc. This happens because Delta Encoding eliminates some patterns from the data.
    - In order to avoid the impact a minimum match distance constraint can be used when doing similarity matching.
    - Two chunks are only considered for Delta Encoding if their min-heap hashes match and they are physically separated from each other by at least a given number of bytes within the data stream.
    - Compression algorithms look for patterns within a given window limited by their dictionary size. So the minimum match distance should be a value greater than this dictionary size in bytes. This avoids any negative impact on the final compression ratio.

The following figure shows the flowchart for the entire process.