- **Top-K Problem Definition:** Finding the top-K most frequent elements from a data stream.
- **Top-K Problem Examples:** Finding the top-K searched terms on Google, the top-K viewed YouTube videos, or the top-K songs on Spotify.
- **Top-K Problem Challenges:** The sheer volume of incoming data makes it difficult to determine the top-K elements in real-time.
- **Event Storage:** Store every event with its timestamp to support arbitrary time range queries.
- **User Experience Priority:** Prioritise fast user interactions, including event publishing and leaderboard reads.
- **Trade-off Between Precision and Speed:** Precise results are slower to compute, while approximations offer faster reads but less accuracy.
- **Event Ingestion and Storage:** Events are ingested by multiple horizontally scaled event publisher servers, then stored in Kafka, ideally partitioned by event ID (e.g., song ID, video ID, search term).
- **Data Aggregation and Processing:** Spark Streaming aggregates events over a defined windowed interval (e.g., hourly) and exports them to HDFS.
- **Data Format and Compression:** Events are converted into Parquet files for storage in HDFS. Parquet's column-oriented structure enables efficient column-level reads and compression, particularly for timestamp columns. Additionally, Parquet provides useful metadata like row count.
- **Spark Streaming Goal:** Ensure every event of the same type goes to the same Spark streaming node for eventual export to HDFS and Parquet.
- **Counting Phase:** Determine the number of times each event occurred within a specified time range (e.g., last Monday to this Monday).
- **Counting Phase Implementation:** Each Hadoop node calculates local event counts, shuffles data to aggregate counts across nodes, and combines counts from different time ranges.
- **Data Aggregation and Shuffling:** Aggregating data by event ID and shuffling it to ensure all relevant keys reside on the same node for efficient processing.
- **Caching Mechanism for Performance Optimisation:** Exploring the possibility of caching counts for recent time periods (e.g., last 7 days) to speed up data reads, although it might introduce complexities in time range logic.
- **Identifying Top K Counts:** After aggregating and shuffling data, the next step is to determine the top K counts for each event within the specified date range.
- **Min-Heap for Top-K Elements:** Using a Min-Heap to efficiently find the top-K counts for each key, reducing time complexity from n log n to n log K.
- **Min-Heap Example:** Illustrating the Min-Heap process with an example of finding the top three counts for Jordan, Tech Lead, and Donald.
- **Aggregating Top-K Elements:** After finding the top-K elements on each HDFS node, the next step is to aggregate them, similar to merging sorted lists.
- **Merging Sorted Lists:** Using a Heap with pointers to efficiently merge multiple sorted lists by always selecting the largest element.
- **Heap Operation:** Picking the largest element from the Heap and moving the corresponding pointer in the sorted list.
- **Performance Bottleneck:** Counting event occurrences for each query range is slow, especially for large datasets.
- **Time Range Granularity:** Restricting queries to specific hour-long intervals allows for caching top-k results for each hour.
- **Approximate Solution for Top-k Aggregation:** Combining top-k results from smaller time windows provides an efficient but potentially inaccurate approximation for larger time ranges.
- **Cache Utilisation for Efficiency:** Leveraging cached top-k results from previous intervals avoids recalculating counts for every event, speeding up queries.
- **Windowed Top-K Calculation:** Using exact top-k solutions from each hour interval.
- **Processing Options:** Batch processing for hourly top-k calculation and storage in a time series database, or stream processing for faster results.

- **Stream Processing Implementation:** Kafka for event ingestion and sharding, Spark Streaming for mini-batch processing and local top-k computation, and Kafka again for publishing results.
- **Data Ingestion and Processing:** Using Spark Streaming to consume data and Flink to merge sorted lists and store them in a time series database.
- **Synchronisation Mechanism:** Leveraging ZooKeeper to track the number of active Spark Streaming consumers and ensure all local top-K lists are received before merging.
- **Data Retrieval:** Clients can query the leaderboard service for top-N events over a specific time period, triggering the service to fetch and aggregate data from the time series database.
- **Data Aggregation and Processing:** The process involves aggregating data from three time series database partitions, potentially on the same node for efficiency. The data is partitioned by time intervals and source.
- **Leaderboard Server Operations:** The leaderboard server utilises a hashmap to aggregate total counts for events and a min-heap to determine the top K events.
- **Optimising Data Flow:** To expedite data processing, the bottleneck of coordinating and combining local counts from Spark streaming consumers can be addressed by aggregating all counts on a single node for direct publishing to the time series database.
- **Count Min Sketch Algorithm:** An algorithm used to approximate counts with a fixed memory size, addressing the challenge of handling billions of unique events.
- **Hash Functions and Buckets:** The algorithm uses multiple hash functions to map events to a fixed-size buffer, represented as a 2D matrix with buckets.
- **Event Processing and Counting:** As events are processed, their corresponding hash values determine the buckets they fall into, allowing for approximate counting.
- **Approximating Element Count:** Using the minimum value from buckets to approximate the count of elements like ‘a’ and ‘b’.
- **Accuracy of Approximation:** Acknowledging that the approximation might deviate from the actual count as more elements are processed.
- **Handling High Volume Events:** Discussing the challenge of processing a high volume of events, using Google’s search terms as an example, and considering the capacity of Kafka and the count/sketch server.
- **Goal of Optimisation:** To achieve the fastest possible processing speed by minimising contention, specifically lock contention on the Count-Min Sketch server.
- **Contention Mitigation Strategy:** Utilise multiple threads, each dedicated to a single Kafka partition, to independently pull data, update the Count-Min Sketch, and then aggregate results hourly.
- **Atomic Integer Usage and Locking:** Employ atomic integers for maintaining data integrity during concurrent updates, acknowledging the inherent locking mechanism involved.
- **Count Min Sketch Design:** Each thread maintains its own Count Min Sketch to avoid contention when reading from Kafka partitions.
- **Data Aggregation and Export:** Every hour, all threads send their Count Min Sketches to a merging thread, which aggregates them and exports to the time series database.
- **Fault Tolerance:** State machine replication is used to ensure that the system can recover from failures.
- **State Machine Replication for Redundancy:** Implementing a secondary Count-Min Sketch server (CMS2) to replicate the state of the primary server (CMS1) for fault tolerance.
- **Event Publishing and Aggregation:** Publishing events to Kafka for fast processing, then aggregating approximate counts hourly in the Count-Min Sketch server and exporting them to a time series database with a replica.
- **Data Flow and Processing:** Kafka events, sharded by event ID, are processed by the Count-Min Sketch server for aggregation and also fed into Spark Streaming for further analysis.
- **Data Ingestion and Processing:** Spark Streaming consumers ingest data from AFA partitions, cache event counts, and compute local top-K values.
- **Data Aggregation and Storage:** Local top-K values are exported to Kafka, and a Flink instance aggregates them using ZooKeeper to determine the number of Spark Streaming nodes.

- **Exact Query Resolution:** For precise queries, parquet files with event counts and timestamps are generated hourly and queried using Spark to provide exact top-K results.
- **Server Scalability:** Discussing the challenges of handling massive event streams from major applications on a single server.
- **Performance Optimisation:** Highlighting the need for extremely fast servers to process events efficiently and avoid contention.
- **Content Conclusion:** Wrapping up the video and thanking the viewers.