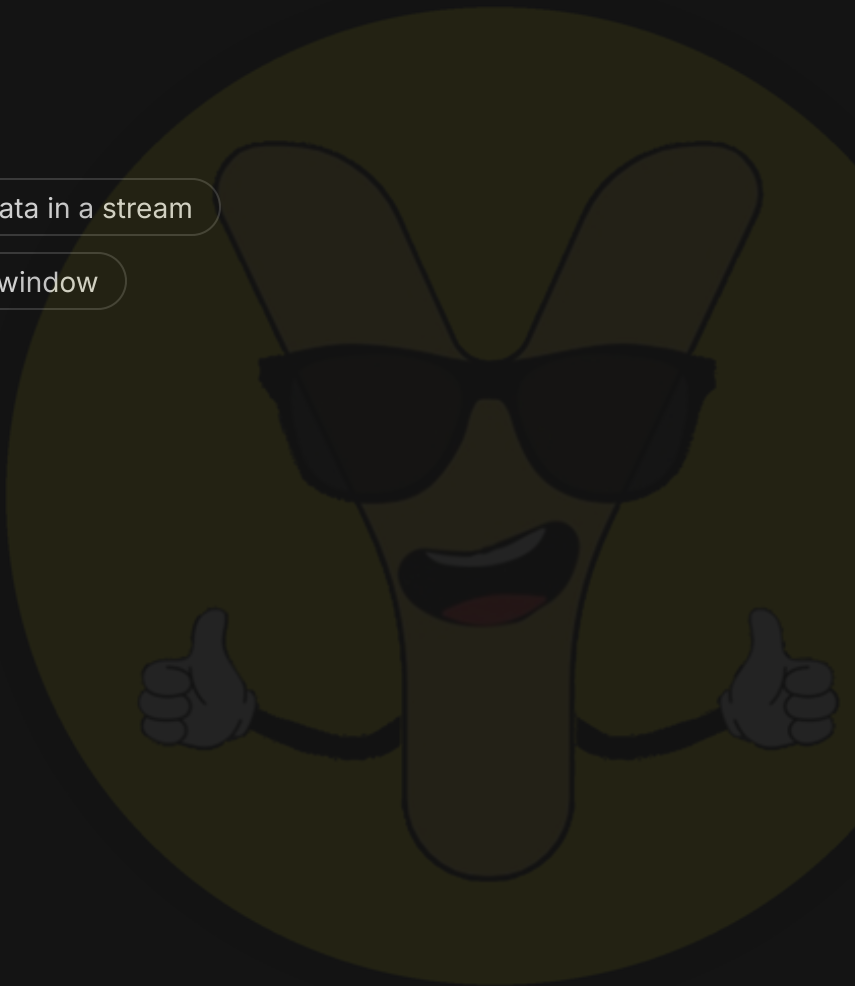
AKTU · 3RD YEAR · SEMESTER 6 · BAD601

Data Analytics

Unit 3 — Mining Data Streams

- Introduction to streams concepts
- Stream data model and architecture
- Stream computing
- Sampling data in a stream
- Filtering streams
- Counting distinct elements in a stream
- Estimating moments
- Counting oneness in a window
- Decaying window
- Real-time Analytics Platform (RTAP) applications
- Case studies: real time sentiment analysis, stock market predictions



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

Introduction to Data Streams

DATA STREAM

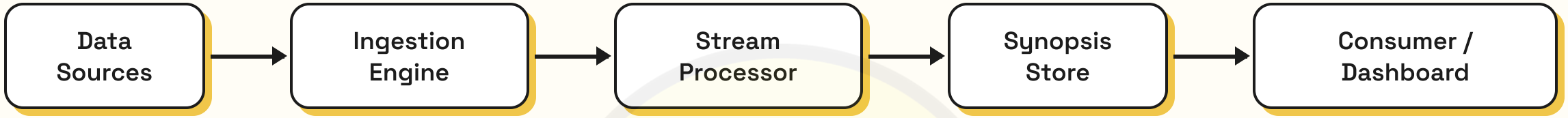
A data stream is a continuous, real-time, ordered sequence of data items arriving at high velocity where the underlying system cannot store the entire dataset in main memory or disk.

- Data items arrive continuously and unbounded over time.
- Limited storage demands sub-linear memory and single-pass processing algorithms.
- Queries must be answered in real-time or near real-time with low latency.
- Stream elements cannot be re-examined unless explicitly cached in a small window.

Batch Processing vs. Stream Processing

Feature	Batch Processing	Stream Processing
Data Scope	Queries over entire static dataset	Queries over sliding window or real-time tuple
Data Volume	Bounded, finite dataset size	Unbounded, continuous infinite flow
Latency	Minutes to hours (High latency)	Milliseconds to seconds (Low latency)
Access Pattern	Random or sequential multi-pass access	Single-pass sequential access only
Storage	Disk-intensive (HDFS, Relational DB)	In-memory or low-overhead volatile memory
Computation	MapReduce, Spark Batch	Apache Flink, Spark Streaming, Storm

Data Stream Processing Architecture



Architectural flow of continuous data stream ingestion, real-time processing, synopsis maintenance, and output delivery.

Stream Computing Principles and Challenges

Stream computing models process data directly in-memory as tuples arrive without persistent disk writes.

- Queries are standing queries executed continuously as new data tuples arrive.
- Ad-hoc queries must operate on summarized stream synopses rather than raw data.
- Main memory is insufficient to hold historical streams, requiring approximate algorithms.
- Concept Drift: Statistical properties of the data stream change unpredictably over time.
- Out-of-order and delayed tuple arrivals occur due to network transmission latency.

Sampling Data in Data Streams

RESERVOIR SAMPLING

Reservoir Sampling is a randomized algorithm used to select a uniform random sample of size s from an unbounded stream of unknown size n in a single pass.

- Guarantees every element in the stream has an equal probability s/n of inclusion.
- Maintains a fixed-size reservoir array of size s in memory at all times.
- Prevents sample bias when total stream length is not known in advance.

Reservoir Sampling Algorithm and Numerical Example

Maintain a sample of size $s = 10$. Compute the probability of the 100th stream element N_{100} being retained in the reservoir.

1. Step 1: Store the first $s = 10$ stream elements directly in the reservoir array.
2. Step 2: For the $n\text{th}$ element where $n > 10$, generate a random integer $k \in [1, n]$.
3. Step 3: If $k \leq 10$, replace the element at index k in the reservoir with the new element.
4. Step 4: Probability N_{100} is initially selected is $p_1 = \frac{10}{100} = 0.10$.
5. Step 5: Probability N_{100} survives when element N_{101} arrives is $1 - (\frac{10}{101} \times \frac{1}{10}) = \frac{100}{101}$.

Every element up to stream length n remains in the reservoir with uniform probability $p = \frac{s}{n}$.

Filtering Streams: Bloom Filters

BLOOM FILTER

A Bloom Filter is a space-efficient probabilistic data structure used to test set membership in a stream, allowing false positives but zero false negatives.

- Consists of a bit array of size m initialized to 0s and k independent hash functions.
- Each hash function maps a stream element to an index in $[0, m-1]$.
- Lookup returns YES (element might be in set) or NO (element is definitely not in set).
- Items can be added to the set, but items can never be removed from a standard Bloom filter.

Bloom Filter Probability & Parameter Optimization

Given a bit array of size $m = 1000$ bits, $n = 100$ inserted elements, and $k = 5$ hash functions, estimate the false positive probability.

1. Step 1: Probability a specific bit remains 0 after 1 hash operation is $(1 - \frac{1}{m})$.
2. Step 2: Probability bit is 0 after inserting n items with k hash functions is $(1 - \frac{1}{m})^{kn} \approx e^{-kn/m}$.
3. Step 3: False positive probability formula: $P_{fp} = (1 - e^{-kn/m})^k$.
4. Step 4: Compute exponent: $\frac{kn}{m} = \frac{5 \times 100}{1000} = 0.5$.
5. Step 5: Calculate $P_{fp} = (1 - e^{-0.5})^5 = (1 - 0.6065)^5 = (0.3935)^5 \approx 0.0094$.

The false positive rate is approximately 0.94%. Optimal hash functions count is $k = \frac{m}{n} \ln 2$.

Counting Distinct Elements: Flajolet-Martin Algorithm

FLAJOLET-MARTIN (FM) ALGORITHM

The Flajolet-Martin algorithm estimates the number of distinct elements in a data stream in a single pass using logarithmic memory space.

- Hashes stream elements to uniform binary integers of length L.
- Tracks $r(x)$: the number of trailing zeros in the binary representation of hash $h(x)$.
- Maintains $R = \max(r(x))$ across all stream items seen so far.
- Estimates distinct elements count as 2^R, or $2^R/\phi$ where $\phi = 0.77351$.

Flajolet-Martin Algorithm Worked Example

Trace the FM algorithm for a stream yielding hash outputs: {0110, 1000, 0100, 1100, 0010}. Estimate the distinct element count.

1. Step 1: Count trailing zeros $r(x)$ for each hashed stream element:
2. Hash 0110: trailing zeros $r = 1$.
3. Hash 1000: trailing zeros $r = 3$.
4. Hash 0100: trailing zeros $r = 2$.
5. Hash 1100: trailing zeros $r = 2$.
6. Hash 0010: trailing zeros $r = 1$.
7. Step 2: Find maximum trailing zero count: $R = \max(1, 3, 2, 2, 1) = 3$.
8. Step 3: Compute distinct element estimate: $F_0 = 2^R = 2^3 = 8$.

Estimated distinct count is $2^3 = 8$. Accuracy is improved by averaging across multiple hash function groups.

Estimating Moments in Streams: AMS Algorithm

STREAM MOMENTS & AMS ALGORITHM

The k-th moment of a stream is defined as $M_k = \sum_{i=1}^{v} (m_i)^k$, where m_i is the frequency of element i. The AMS algorithm estimates these moments in limited space.

- 0-th Moment M_0: Number of distinct elements in the stream.
- 1-st Moment M_1: Total length of the stream (Total tuples $N = \sum m_i$).
- 2-nd Moment M_2: Surprise Number, measuring variance and skewness of data distribution.
- AMS (Alon-Matias-Szegedy) algorithm provides an unbiased estimator for M_2.

AMS Algorithm for Second Moment Estimation

Calculate the exact second moment M_2 for a stream sequence: A, B, A, C, B, A, B, A.

1. Step 1: Compute frequencies of each distinct item in stream length $N = 8$:
2. Frequency of A: $m_A = 4$
3. Frequency of B: $m_B = 3$
4. Frequency of C: $m_C = 1$
5. Step 2: Apply formula $M_2 = \sum (m_i)^2$.
6. Step 3: Calculate $M_2 = (4)^2 + (3)^2 + (1)^2 = 16 + 9 + 1 = 26$.
7. Step 4: AMS Estimator: Select random position X, count occurrences from X to end as c. Value $e = N(2c - 1)$.

Exact Surprise Number $M_2 = 26$. AMS estimator averages multiple variables e to approximate M_2.

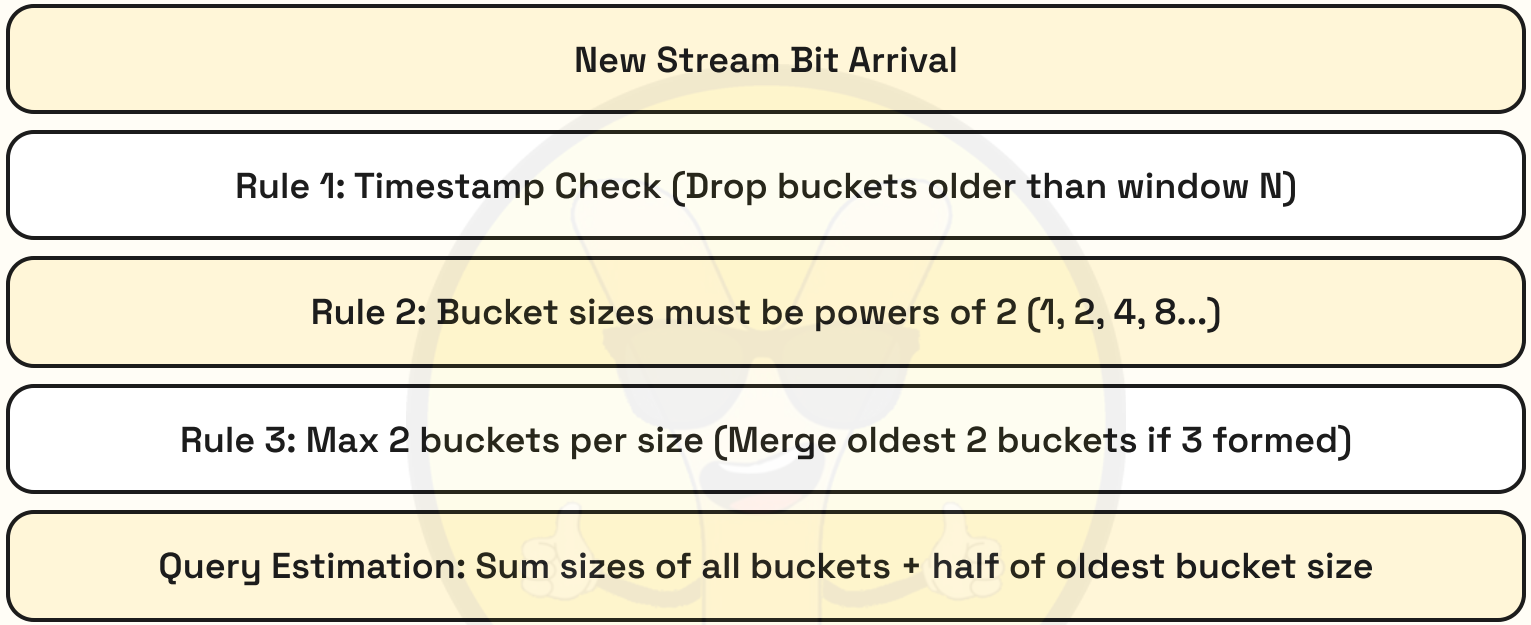
Counting Oneness in a Window: DGIM Algorithm

DGIM ALGORITHM

The Datar-Gionis-Indyk-Motwani (DGIM) algorithm estimates the number of 1s in a sliding window of size N using $O(\log^2 N)$ space with a maximum error bound of 50%.

- Solves the problem of counting 1s when storing the entire window of size N is impossible.
- Divides the window into buckets containing power-of-two number of 1s: $1, 2, 4, 8, \ldots$
- Bucket sizes are exponentially increasing to strictly bound memory usage.
- Timestamps are recorded modulo N to maintain window boundaries efficiently.

DGIM Bucket Structure and Maintenance Rules



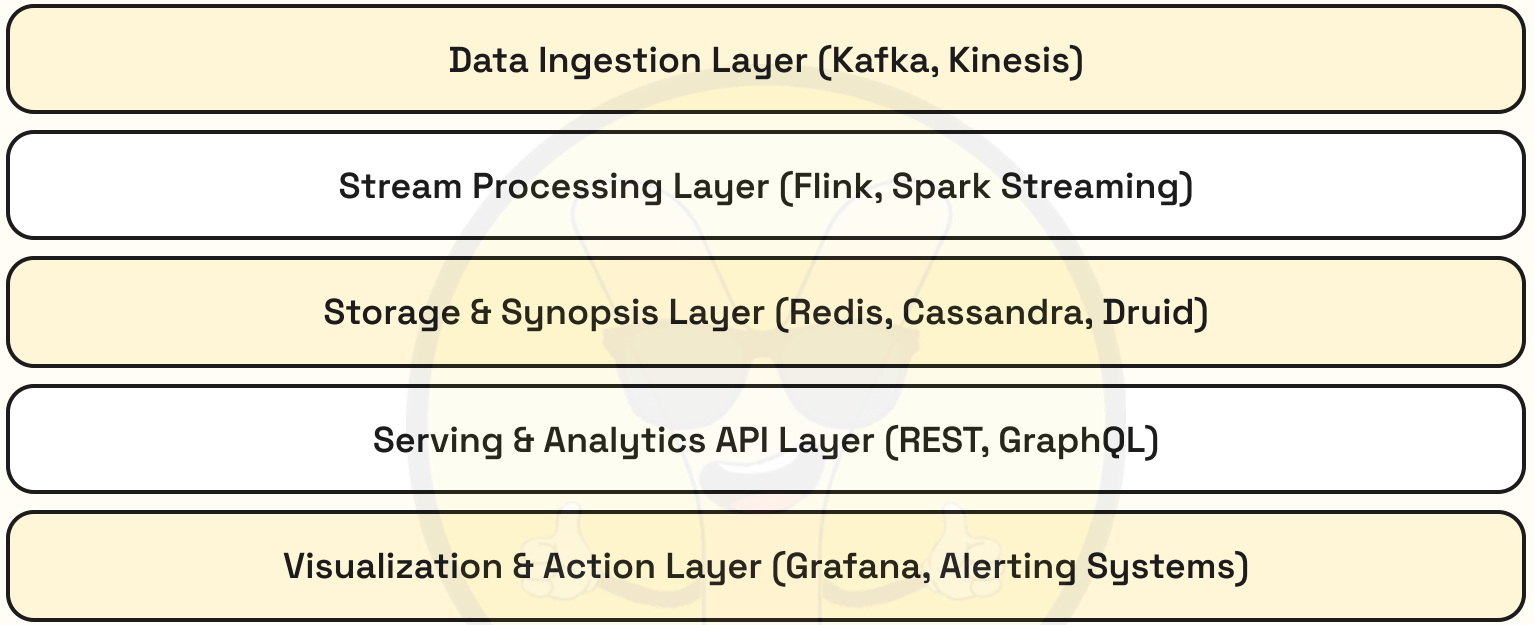
Processing steps and structural constraints for DGIM sliding window stream management.

Decaying Windows Concept and Exponential Weighting

Decaying windows assign exponentially higher weights to recent elements while gradually discounting older data.

- Addresses continuous streams where recent events are more significant than historical events.
- Decay factor parameter c controls attenuation rate (e.g., $c = 10^{-5}$).
- Smooth decay function: Weight of item arriving t steps ago is $f(t) = e^{-ct}$.
- Sum updating rule: When new element x arrives, update sum $S_{new} = S_{old} \times (1 - c) + x$.
- Requires zero historical storage, computing running weighted sum efficiently in $O(1)$ space.

Real-Time Analytics Platform (RTAP) Architecture



Layered architecture of enterprise Real-Time Analytics Platforms (RTAP).

Case Study 1: Real-Time Sentiment Analysis

Processing live social media and customer feeds to track evolving brand sentiment continuously.

- Data Ingestion: Continuous streaming of tweets/reviews via Apache Kafka into stream processing framework.
- Filtering & Preprocessing: Remove stop-words; apply Bloom filters for duplicate tweet detection.
- Sliding Window Analysis: Compute sentiment score per 5-minute sliding window.
- Decaying Window Application: Exponentially decay past sentiment scores to detect sudden public reaction shifts.
- Alerting: Real-time trigger generation when negative sentiment ratio exceeds defined threshold.

Case Study 2: Real-Time Stock Market Predictions

Analyzing high-frequency financial tick data for instantaneous trading strategies.

- Data Input: High-frequency trade ticks arriving at sub-millisecond intervals.
- Feature Extraction: Real-time calculation of Moving Averages, RSI, and Volume Spikes using sliding windows.
- Moment Estimation: AMS algorithm monitors volatility (second moment) across stock tickers.
- Anomaly Detection: Filtering out outlier transactions via stream synopsis structures.
- Low-Latency Execution: Trigger automated buy/sell orders in sub-millisecond timeframes.

Comparison of Data Stream Algorithms

Algorithm	Primary Task	Space Complexity	Key Property / Bound
Reservoir Sampling	Uniform random sample	O(s) sample size	Exact uniform distribution
Bloom Filter	Stream membership test	O(m) bit array	Zero false negatives
Flajolet-Martin	Distinct elements count	O(log N)	Probabilistic F0 estimate
AMS Algorithm	Moments estimation (M2)	O(log N)	Unbiased surprise number
DGIM Algorithm	Count 1s in window	O(log^2 N)	Max 50% relative error
Decaying Window	Weighted aggregation	O(1) memory	Exponential weight e^{-ct}

AKTU Semester Exam Questions - Unit 3

- **Q1** Differentiate between batch processing and stream processing models with suitable examples.
- **Q2** Explain Reservoir Sampling algorithm. Prove that every element in a stream has an equal probability of selection.
- **Q3** Describe the working of Bloom Filter. Derive the expression for its false positive probability.
- **Q4** Explain the Flajolet-Martin (FM) algorithm for counting distinct elements in a data stream with a numerical example.
- **Q5** Define moments of a stream. Describe how the AMS algorithm estimates the second moment (Surprise Number).
- **Q6** Explain the DGIM algorithm for counting oneness in a window of size N. State its space complexity and bucket rules.
- **Q7** What is a Decaying Window? Explain how exponential decay weighting is computed without storing past data.
- **Q8** Design a high-level Real-Time Analytics Platform (RTAP) architecture for a live stock market prediction engine.

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

