Streams management

Knowledge objectives

1. Recognize the relevance of stream management
2. Recognize the elements in a log
3. Enumerate the most relevant characteristics of streams
4. Recognize the importance of the lambda architecture
5. Explain to which extent a DBMS can manage streams
6. Explain the parameters of the storage layer

Understanding Objectives

1. Decide the probability of keeping a new element or removing an old one from memory to keep equi-probability on load shedding
2. Decide the parameters of the hash function to get a representative result on load shedding
3. Decide the optimum number of hash functions in a Bloom filter
4. Approximate the probability of false positives in a Bloom filter
5. Calculate the weighted average of an attribute considering an exponentially decaying window

Definition

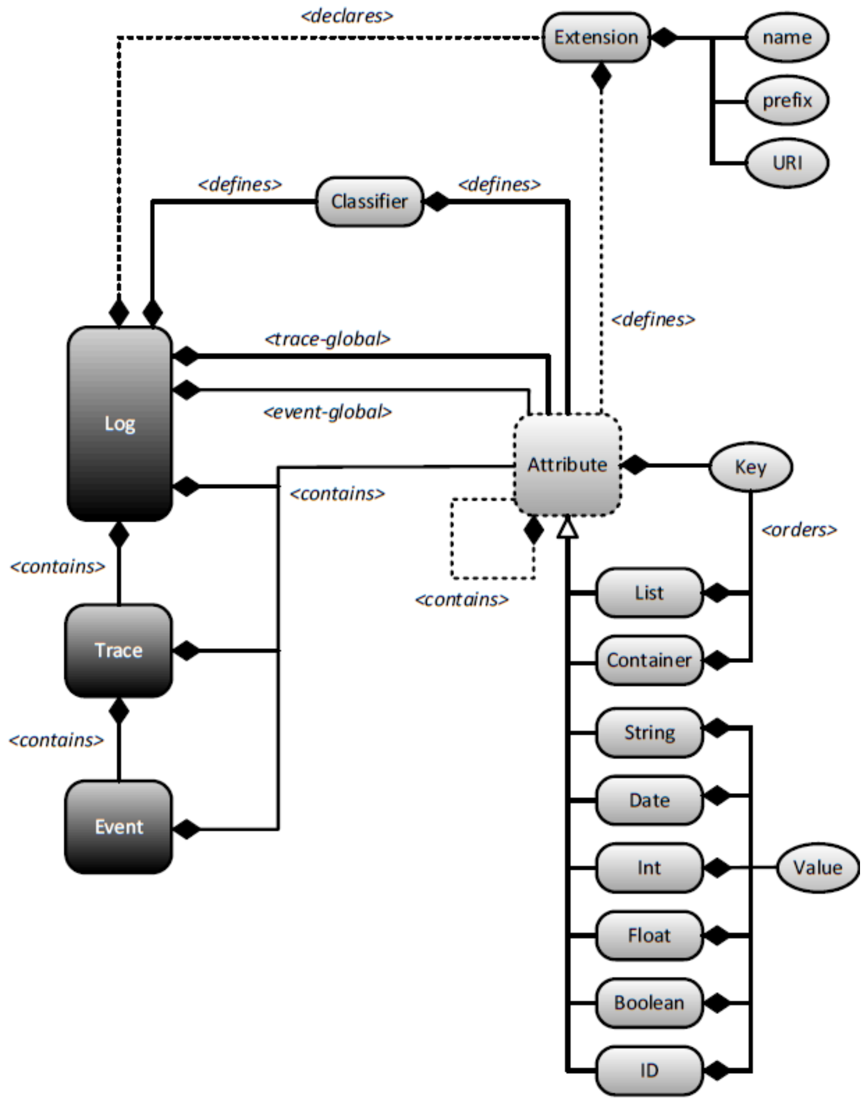
“Class of software systems that deals with processing streams of high volume messages with very low latency.”

Michael Stonebraker, Encyclopedia

Tens of thousands of elements per second

- Internet traffic analysis
- Trading on Wall Street
- Fraud detection (i.e., credit cards)
- High-way traffic monitoring
- Surveillance cameras
- Command and control in military environments
- Log monitoring
  - Google receives several hundred million search queries per day
- Click analysis
  - Yahoo! Acceps billions of clicks per day
- Scientific data processing (i.e., sensor data)
  - One million sensors reporting at a rate of ten per second would generate 3.5TB/day (only 4 bytes per message)
- RFID monitoring
  - Venture Development Corporation predicted in 2006 that RFID can generate in Walmart up to 7TB/day (≈ 292GB/hour ≈ 5GB/minute ≈ 80MB/second)

Extensible Event Stream (XES)



http://www.xes-standard.org
Eindhoven University of Technology

Stream characterization

- Arrival rate not under the control of the system
  - In general, it is faster than the processing time
    - Algorithms must work with only one pass of the data
- Unbounded memory requirements
  - Some drastic reduction is needed
- Keep the data moving
  - Only volatile storage
- Support for real-time application
  - Latency of 1 second is unacceptable
    - Need to scale and parallelize
- Arrival order not guaranteed
  - Some data may be delayed
- Imperfections must be assumed
  - Some data will be missing
- There is temporal locality
  - Data (characteristics) evolve over time
- **Approximate** (not accurate) **answers** are acceptable
  - Outcomes must still be predictable

Databases vs Streams

	Database management	Stream management
Data	Persistent	Volatile
Access	Random	Sequential
Queries	One-time	Continuous
Support	Unlimited disk	Limited RAM
Order	Current state	Sorted
Update rate	Relatively low	Extremely high
Temporal requirements	Little	Real-time
Accuracy	Exact data	Imprecise data
Heterogeneity	Structured data	Imperfections
Algorithms	Multiple passes	One pass

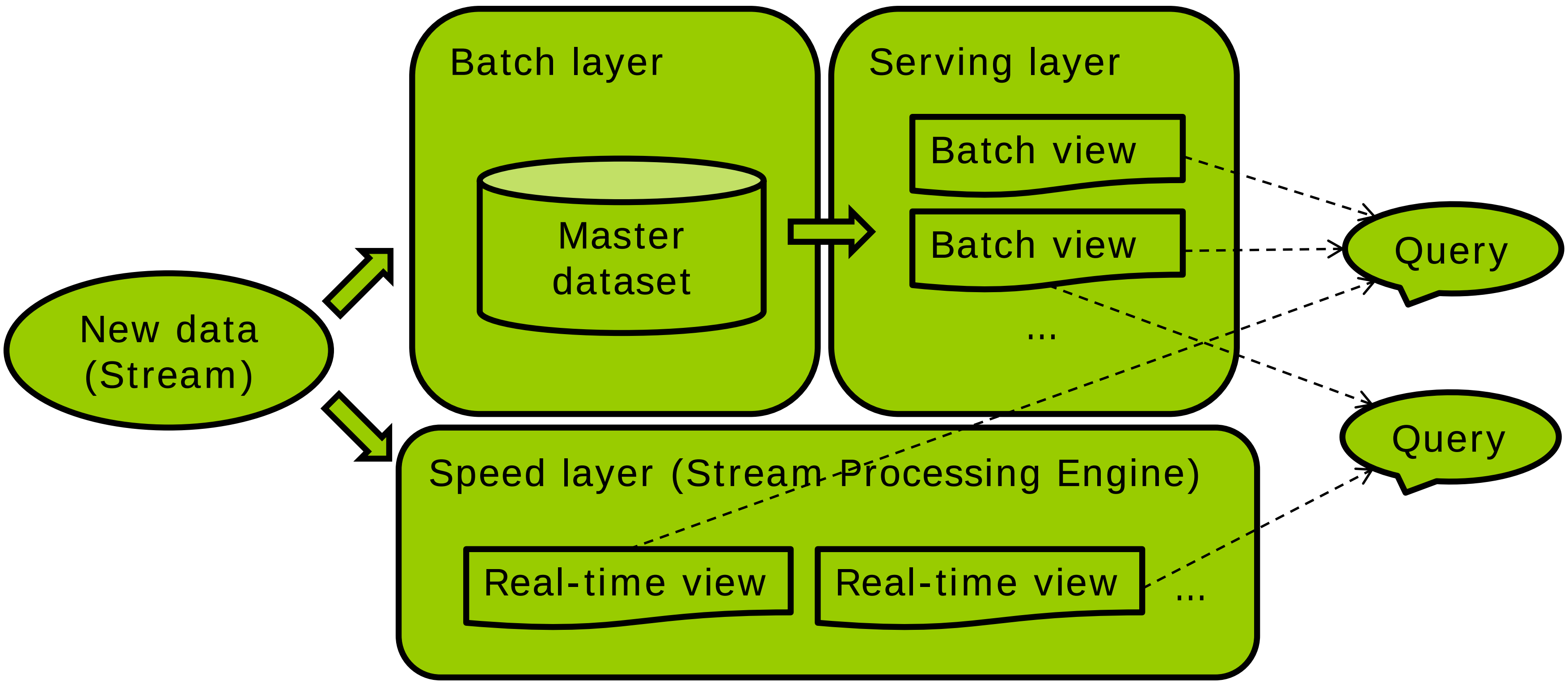
Kinds of queries

- Depending on the trigger
  - Standing
  - Ad-hoc
- Depending on the output
  - Alerts
  - Result set
- Depending on the inputs
  - Based on the last element
  - Based on the X last elements
    - Sliding window
  - Based on a summary
    - Synopsis

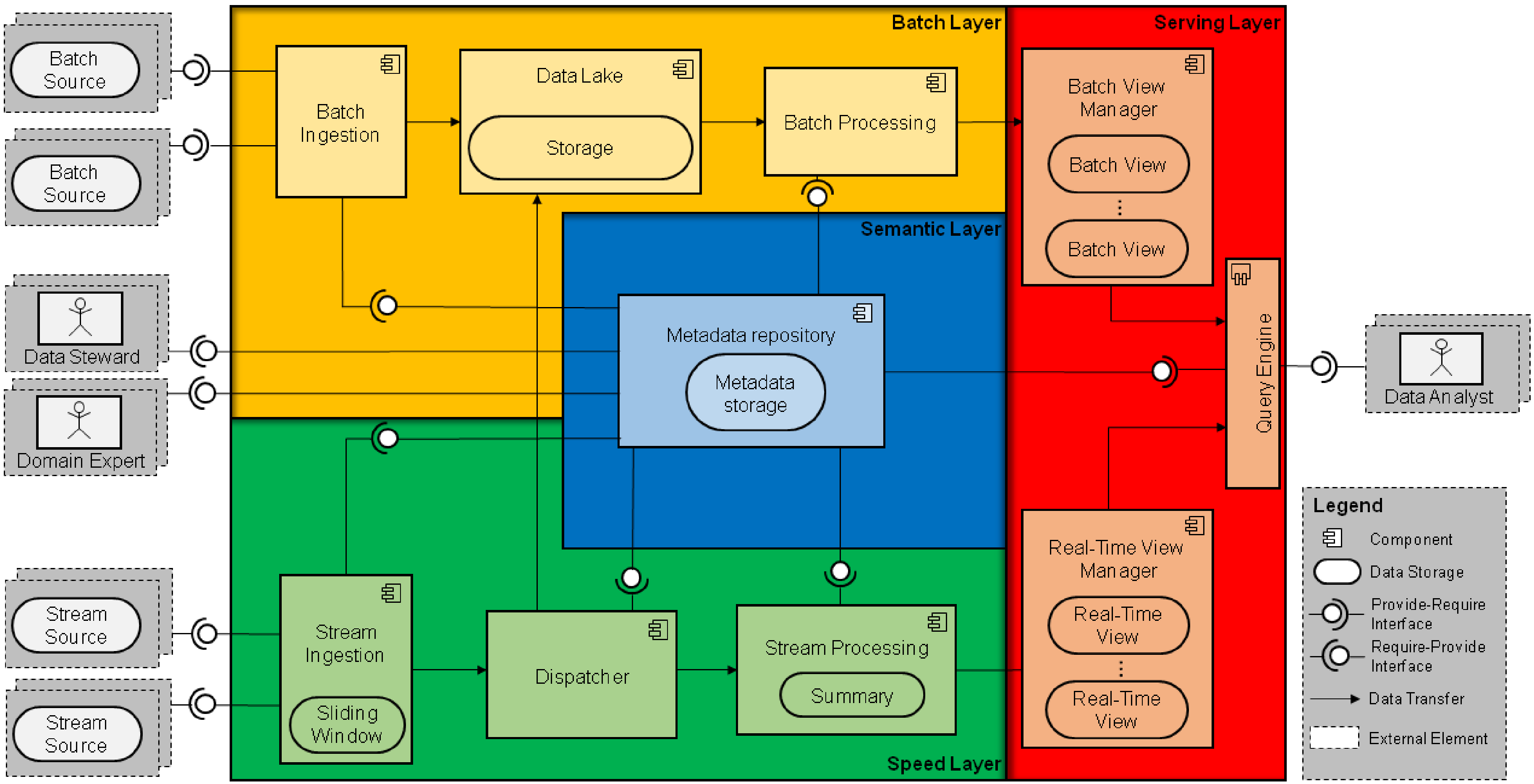
Architectural patterns for near-real time

- Stream ingestion
- Near-real time event processing
  - Non-partitioned
    - Get profile information needed for decisions
    - Requires nearly no coding beyond the application-specific logic
  - Partitioned
    - Define a key to partition data
      - Match incoming data to the subset of the context data that is relevant to it
- Complex topology
  - Aggregation
  - Machine learning

Lambda architecture



Bolster software reference architecture



Oracle 10g temporary table

CREATE GLOBAL TEMPORARY TABLE <tablename> (...) [ON COMMIT {<u>DELETE ROWS</u>|PRESERVE ROWS}];

- Relational mapping
  - Each element is a tuple
  - The sliding window is a relation
- Metadata is in the dictionary
- Data is not persistent
  - Transaction specific
  - Session specific
- Does **not** support:
  - Foreign keys
  - IOT
  - Cluster
  - Partitions
  - Parallelism

Storage (queue) manager parameters

- Architectural
  - Meterialization (how data is provided)
    - Active/Passive
  - Access model (how data is retrieved)
    - Push/Pull
- Functional
  - Parameters
    - Synchronization (condition checking)
    - Schema (information parsing)
  - Persistence (storage media)
    - Persistent/Transient
- Performance-related
  - Read access pattern
    - Sequential/Random/Clustered
  - Update access pattern
    - Expiration
      - Never/Ordered/Unordered/Replaced
    - Consumption
      - Never/Ordered/Eager
    - Modification
      - No-update/In-place/Random/FIFO
  - Sharing

Challenges and approaches

- Limited computation capacity
  - Sampling (i.e., Load shedding)
    - Probabilistically drop stream elements
  - Filtering (i.e., Bloom filters)
- Limited memory capacity
  - Sliding window
    - Discard elements
      - Aging (use only most recent data)
  - Exponentially decaying window
    - Weight elements
  - Synopsis
    - Approximate solutions
    - Examples:
      - Concise sampling
        - Works under a limited number of distinct values
      - Histograms
        - Works under uniform distribution of values in a bucket

Load shedding (Keeping equi-probability)

- Mistakes in case of infinite streams:
  a) Fix the values at the beginning
  b) Remove old values from memory
- Goal:
  - All past elements have the same probability of being in memory at any time
- Definitions:
  - Memory positions: *p*
  - Elements seen: *n*
- Solution:
  - Probability of keeping the new element *n+1*
    - p/(n+1)
  - Probability of removing an element from memory
    - 1/p

Load shedding (Statement)

"Select a subset of the stream so that answering ad-hoc queries gives a statistically representative result."

Example: *Given a stream of tuples [user, query, time], we can store 10% of the tuples. If we randomly keep* ***1/10 of the tuples****, then we would get the wrong answer to "Average number of duplicate queries for a user"!!!*

Definitions:

s = *queries issued once*

d = *queries issued twice*

No queries issued more than twice

The sample will contain:

s/10+18d/100 *queries issued once*

d/100 *queries issued twice*

The answer would be:

$d/(10s+19d) \neq d/(s+d)$

Solution:

Keep ***1/10 of the users*** *(use a hash function of the key)*

A. Rajaraman and J. Ullman

Load shedding (Generalization)

- The queries may need different grouping keys or the key can be compound
  - Use the group by set in the hash function
- The memory may be limited
  - Take a hash function to a large number of values and keep only elements mapping to a value bellow *t* (just dynamically reduce *t* as you are running out of memory)
- The system must quickly adjust to varying incoming stream processing rates

Bloom filters (Statement)

"Accept those elements in the stream that meet a criterion (based on looking for membership in a set), others are dropped."

Example: Given an e-mail stream of tuples [address,text], we have a list of 10^9 allowed addresses (20 bytes each) and only 1GB of memory available.

Solution:

Use the memory as an array of bits and map the addresses by means of a hash function (approximately 1/8 bits will be set)

Note:

Some spam will still get through the filter

A. Rajaraman and J. Ullman

Bloom filters (Generalization)

- Elements:
  - A set of *m* key values
  - **One array** of *n* bits
  - A list of *k* hash functions (h_i: key $\rightarrow$ *n*)
- Construction:
  - For each element in the probing set, apply all *k* hash functions and set to 1 the corresponding bits
- Checking:
  - For each element in the stream, apply all *k* hash functions, it will pass only if all corresponding bits are set to 1
- False positives:
  - $(1-e^{-km/n})^k$
- Optimal
  - $k = (n/m)\cdot\ln 2 \rightarrow (1-e^{-km/n})^k = (1/2)^k \approx 0.618^{n/m}$

Bloom filters (Rationale)

- Probability of a bit being set by a hash function
  $1/n$
- Probability of a bit NOT being set by a hash function
  $1-1/n$
- Probability of a bit NOT being set by a hash function of ANY key
  $(1-1/n)\cdot(1-1/n)\cdot \ldots \cdot(1-1/n) = (1-1/n)^m = (1-1/n)^{n(m/n)}$
- A good approximation of $(1-\epsilon)^{1/\epsilon}$ for small ϵ is $1/e$
  $(1/e)^{m/n} = (e^{-1})^{m/n} = e^{-m/n}$
- Probability of a bit set by a hash function of ANY key
  $1-e^{-m/n}$
- Probability of a bit set by ANY hash function of ANY key
  $1-(e^{-m/n})^k = 1-e^{-km/n}$
- Probability of all hash functions finding the bit set
  $(1-e^{-km/n})^k$

Bloom filters (Example)

Key values = $\{IP_1, IP_2\}$

Hash functions = $\{f_1, f_2\}$

Array of bits ⟹ 0 0 1 0 1 0 1 0 0 0

Construction

$f_1(IP_1) = 3$ $f_2(IP_1) = 5$

$f_1(IP_2) = 7$ $f_2(IP_2) = 5$

Checking

IP_3 ⟹ $f_1(IP_3) = 3$ $f_2(IP_3) = 7$ **FALSE POSITIVE!**

Exponentially decaying window (Statement)

"Do not make a distinction between old and young element, but just weight them."

*Example: Find the **currently** most popular movie. We could not keep a window big enough!*

Solution:

Keep one weighted counter per movie

Definitions:

c = *small constant (e.g.,* 10^{-6} *or* 10^{-9}*)*

T = *current time*

$f(t) = a_t$ = *element at time t (or 0 if there is no element)*

$g(T-t)$ = *weight at time T of an item obtained at time t*

X = *time since the last update*

Value:

$\Sigma f(i)\cdot g(T-i) = a_{T-i}(1-c)^i,\ i=0..T-1$

Process:

Multiply the current counter by $(1-c)^X$ *and add* a_t

A. Rajaraman and J. Ullman

Exponentially decaying window (Example)

c=0.5

Counter = 0.28125

Stream

0 1 0 0 1 0 0 ...

Activity

- *Objective: Understand three approaches to handle streams*
- *Tasks:*
  1. *(7') Individually solve one exercise*
  2. *(13') Explain the solution to the others*
  3. *Hand in the three solutions*

- *Roles for the team-mates during task 2:*
  a) *Explains his/her material*
  b) *Asks for clarification of blur concepts*
  c) *Mediates and* ***controls time***

Summary

- Extensible Event Stream
- Lambda architecture
- Storage manager parameters
- Stream management techniques
  - Load shedding
  - Bloom filters
  - Exponentially decaying window

Bibliography

- A. Rajaraman and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011
  - http://www.mmds.org
- C. C. Aggrawal editor. *Data Streams, models and algorithms*. Springer, 2007
- M. Stonebraker et al. *The 8 Requirements of Real-Time Stream Processing*. SIGMOD Record 34(4), 2005
- N. Polyzotis et al. *Meshing Streaming Updates with Persistent Data in an Active Data Warehouse*. IEEE Trans. Knowl. Data Eng. 20(7), 2008
- L. Liu and M.T. Özsu (Eds.). Encyclopedia of Database Systems. Springer, 2009
- I. Botan et al. *Flexible and Scalable Storage Management for Data-intensie Stream Processing*. EDBT, 2009

Resources

- http://www.xes-standard.org
- http://kafka.apache.org
- https://storm.incubator.apache.org
- https://spark.apache.org/streaming