Data Stream Management System

Reza Salehi, Ajay Rawat

Outline

- “An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations”
- Stanford Stream Data Manager (STREAM)
- Difference between DBMS and DSMS
- Optimizations for DSMS
- DSMS vs. TSDB (Time-series database)

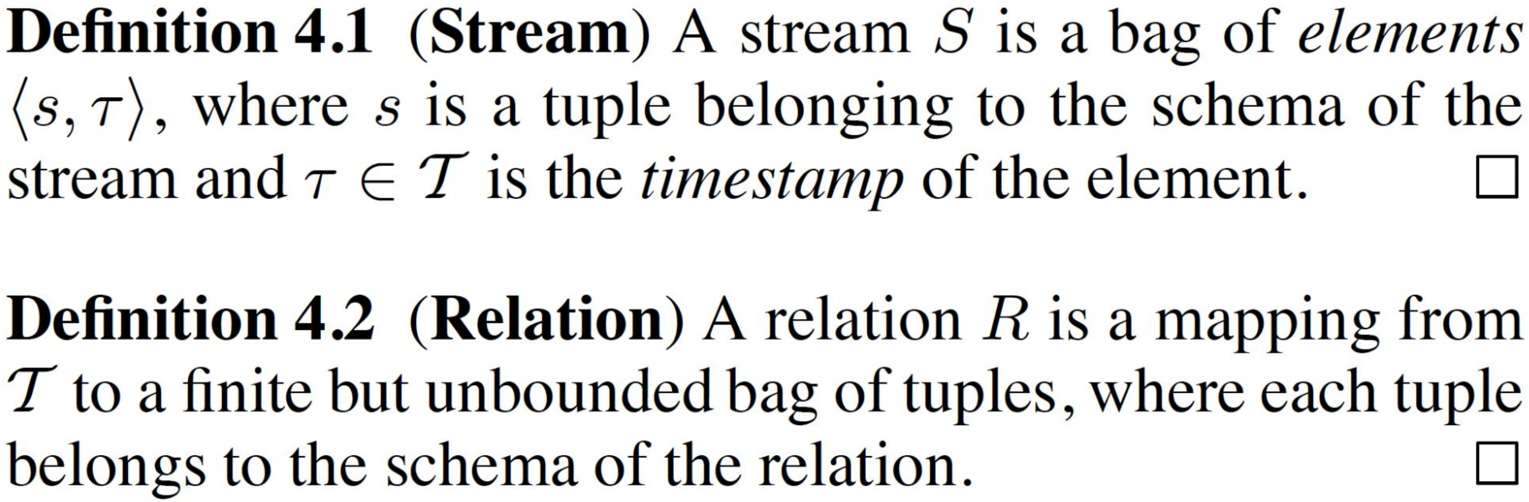
How to represent continuous queries over streams?

- Goal: defining precise semantics for continuous queries over streams.
- Abstract semantics based on three building blocks:
  - Streams and relations
  - Mappings among them
  - Any relational query language to operate on relations
- CQL (for Continuous Query Language) instantiates semantics
- CQL: SQL as the relational query language + extensions

Running example

- Online auction application
- Users, transactions of the users, continuous queries of users or administrators for various monitoring purposes
- Users can register and deregister
- Three kinds of transactions:
    - Open an auction
    - Close an auction they previously started
    - Bid for currently active auctions placed by other users by specifying a bid price
- Monitoring queries: notified about any auction placed by a user from California within a specified price range

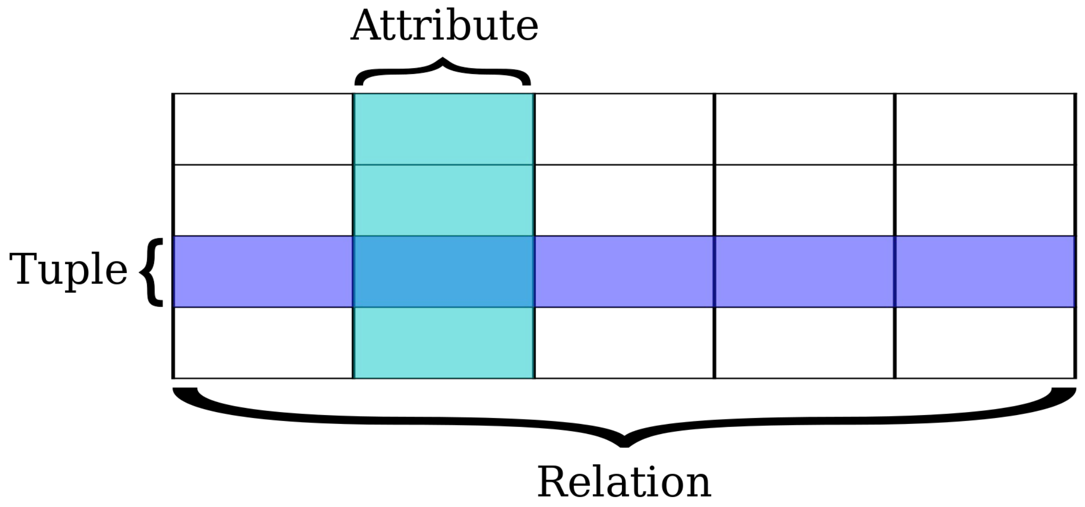
Definitions



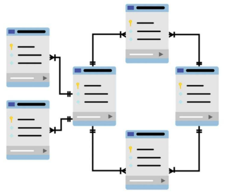
Definition 4.1 (Stream) A stream S is a bag of *elements* $\langle s, \tau \rangle$, where s is a tuple belonging to the schema of the stream and $\tau \in \mathcal{T}$ is the *timestamp* of the element. $\square$

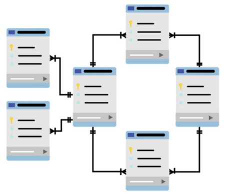
Definition 4.2 (Relation) A relation R is a mapping from $\mathcal{T}$ to a finite but unbounded bag of tuples, where each tuple belongs to the schema of the relation. $\square$

Relational database terminology



Definitions

Streams: Tuples () * Time

Relations: Time $\longrightarrow$ Tuples ()

Note: Timestamps refer to logical times, although it could be physical time.

Definitions

- Source data streams (*base stream*) that arrive at a DSMS + streams that result from queries or subqueries (*derived stream*)
- *Instantaneous relation* denotes relations at a fixed time (Same as ordinary relational databases)

Modeling the running example

Schema of input streams:

- `Register(user id, name)`
- `Deregister(user id)`
- `Open(item id,seller id,start price)`
- `Close(item id)`
- `Bid(item id,bidder id,bid price)`

Mapping operators

- Stream-to-relation: Takes a stream and produces relation with the same schema. At any instant *t*, *R(t)* should be computable from the elements of *S* up to *t*.

- Relation-to-relation: Takes relations $R_1, R_2, \ldots, R_n$ and produces *R*. At any instant *t*, *R(t)* should be computable from $R_1(t), \ldots, R_n(t)$

- Relation-to-stream: takes a relation *R* as input and produces a stream *S* with the same schema as *R* as output. At any instant *t* the elements of *S* with timestamp *t* should be computable from *R(t')* for t' <= t

Mapping operators examples

- “sliding window” over this stream containing the bids over the last ten minutes
- This windowing is an example of stream-to-relation operator
- The output relation *R* at time *t* contains all the bids with timestamp between *t* and *t-10* minutes
- Consider the operator `Avg(bid_price)` over the output relation R
- One single-attribute tuple containing the average price of the bids
- This aggregation is an example of relation-to-relation operator
- Relation-to-stream operator: stream the average price resulting from `Avg(bid_price)` every time the avg. price changes

Abstract semantics for continuous queries

Each query is constructed from three building blocks:

- Any relational query language as a set of relation-to-relation operators
- Window specification language as a set of stream-to-relation operators for extracting tuples from streams (in practice windowing is the most common way of producing relations from streams)
- Relation-to-stream operators: `Istream`, `Dstream`, and `Rstream` (these are expressive enough for almost all queries)

Interaction Among Three Building Blocks

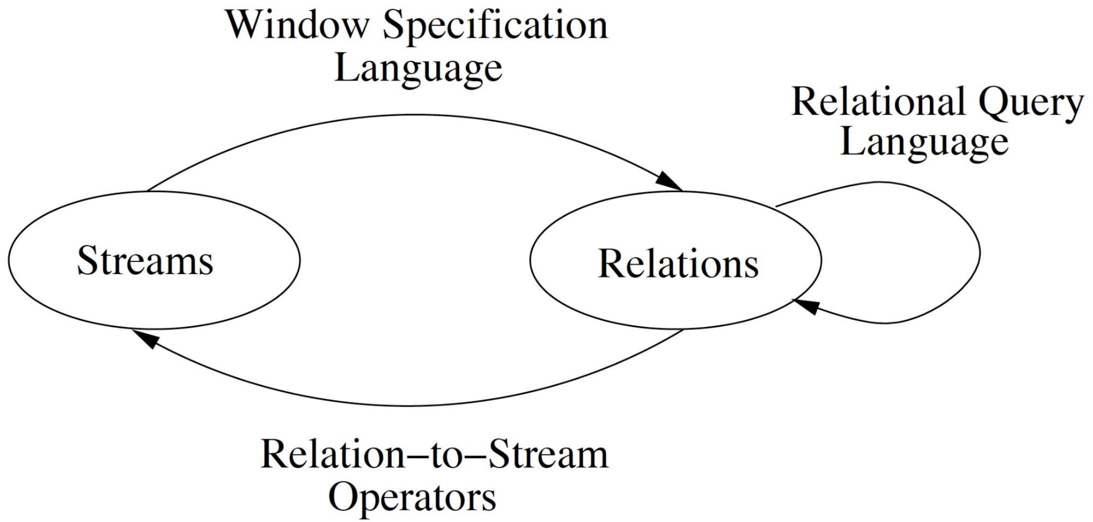

Window Specification Language
Relational Query Language
Streams
Relations
Relation–to–Stream Operators

Figure 2: Mappings used in abstract semantics

Concrete Query Language

CQL: SQL + three syntactic extensions:

1. Anywhere a relation may be referenced in SQL, a stream may be referenced in CQL.
2. Every reference to a base stream, and every subquery producing a stream, must be followed immediately by a window specification.
3. Reference to a relation, or any subquery producing a relation, may be converted into a stream by applying any of the operators `Istream`, `Dstream`, or `Rstream`

Relation-to-stream Operators

- `Rstream` (for relation stream): applied to relation R contains a stream element <s, t> whenever tuple s is in R at time t. Formally,

$$\texttt{Rstream}(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\})$$

Relation-to-stream Operators

- `Istream` (for insert stream): Applied to relation *R* contains a stream element <*s*, *t*> whenever tuple *s* is in R(t) - R(t-1).

$$\texttt{Istream}(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\})$$

Note: R(-1) = ϕ

Relation-to-stream Operators

- `Dstream` (for insert stream): Applied to relation *R* contains a stream element <*s*, *t*> whenever tuple *s* is in R(t-1) - R(t).

$$\texttt{Dstream}(R) = \bigcup_{\tau>0} ((R(\tau-1) - R(\tau)) \times \{\tau\})$$

Note: R(-1) = ϕ

Window Specification Language

CQL supports three types of sliding windows:

1. Time-based: defines output by sliding a window of size *T* over *S*. The relation `S[Range T]` is defined as:

$$R(\tau) = \{s \mid \langle s, \tau' \rangle \in S \ \wedge \\ (\tau' \leq \tau) \ \wedge \ (\tau' \geq \max\{\tau - T, 0\})\}$$

Special case: `S[Now]` where T=0

Window Specification Language

2. Tuple-based: defines output by sliding a window of size *N* tuples over *S*. Relation `S[Rows N]` consists of tuples obtained from the *N* elements with the largest time stamps in *S* no greater than *t* for each *t*.

3. Partitioned-windows which we do not discuss...

Example Queries

Ex 1. *Select auctions where the starting price exceeds 100 and produce the result as a stream.*

```
Select * From Open Where start_price > 100
```

Note: `Istream` operator can convert the output relation into a stream consisting of each element of `Open` that satisfies the filter predicate.

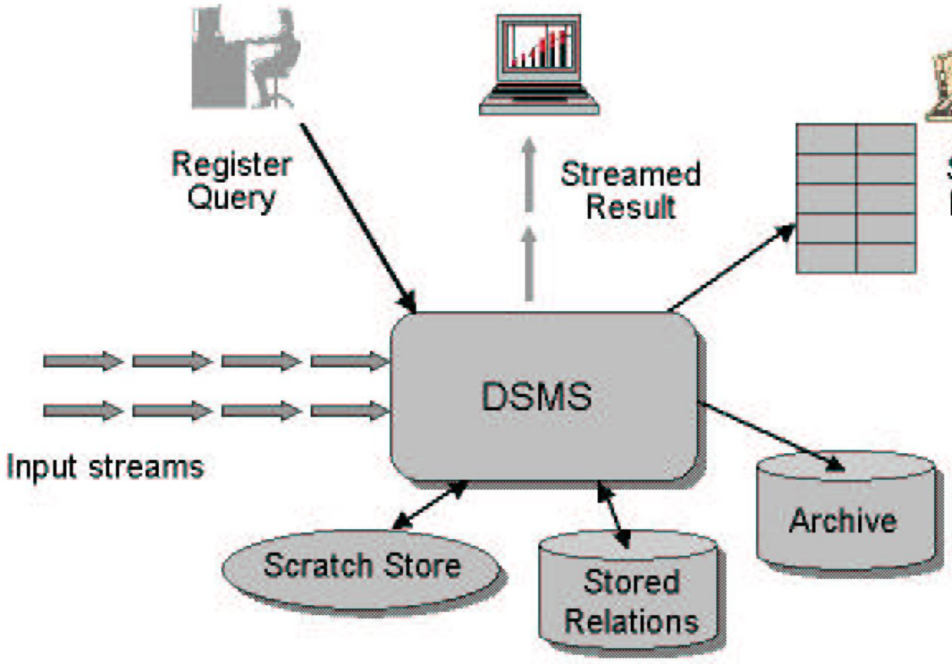
Example Queries

Ex 2. *Maintain a running count of the number of bids in the last hour on items with item id in the range 100 to 200.*

```
Select Count(*) From Bid[Range 1 Hour]
Where item_id >= 100 and item_id <= 200
```

Note: Use `Rstream` If the count should be streamed at each time instant regardless of whether its values has changed. If `Istream` is used it will stream a new value each time the count changes.

Overview of STREAM



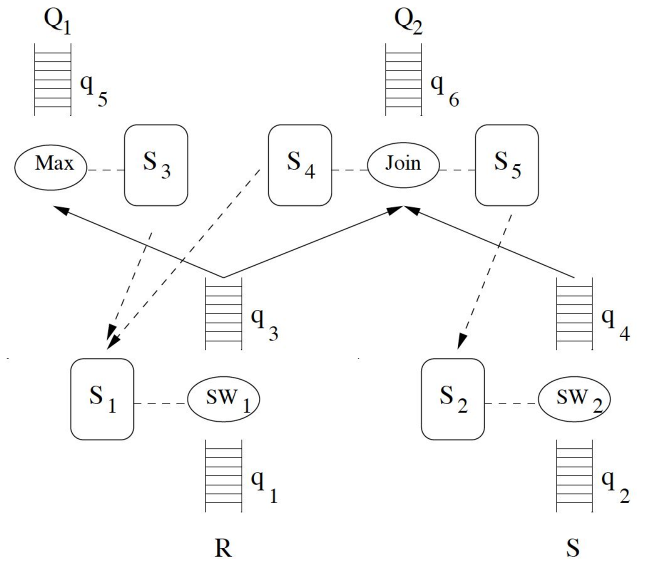
Notes on STREAM

- Incoming *Input Streams* produce data indefinitely
- Intermediate states are stored in *Scratch Store*
- Stream data may be copied to *Archive* for possible offline processing
- Users or applications register *Continuous Queries*
- Queries remain active in the system until they are deregistered
- Results are generally output data streams but could be relational results that are updated over time

Query processing

- Any continuous query registered to the system compiled into a *query plan*
- New query plan is merged into existing query plans whenever possible
- Query plans consists of:
    a. Query operators corresponding to three types of operators discussed previously
    b. Inter-operator queues to buffer the output of one operator passed as input to other operators
    c. Synopses to maintain run-time state associated with operators
- Incoming stream types and relation updates are place in input queues feeding leaf operators
- Synopsis stores intermediate state at some operators in a running query plan needed for future evaluation of that operator
- Most common use case of synopsis: materializing relation or view

Query plan example



- Query Q1 is a windowed-aggregate query
  - Maximum value of attribute R.A over a sliding window
- Q2 is a sliding window join query over R and S
- `SW1`
  - Reads stream R’s tuples from queue q1
  - Updates the sliding window synopsis
  - Outputs inserts & deletes to this sliding window into q1
- `Max`
  - maintains maximum of R.A incrementally over the window using inserts and deletes
  - Whenever the current max expires max needs the entire window rather than the updates -> Materialize the input window in S3
  - S3 is simply a time-shifted version of S1, therefore it can be shared partially with S1
- `SW2` and `Join` are similar....

Operator scheduling

- Global scheduler running in the same thread as all all the operators
- I/O operations are handled by separate thread
- How scheduler works:
  - Scheduler invoked and selects operator to execute
  - Passes the maximum amount of time that the operator should run before returning control
  - Operator may return earlier
- Objectives: minimizing runtime resource consumption, maximizing throughput, careful management of runtime resources
- Irregular rate of arrival, burstiness and variation of data arrival rate over time
- Backlog buffering
- Minimize the memory required for backlog buffering

Operator scheduling

- Chain scheduling:
  - Break up query plans into disjoint chains of consecutive operators that “consume” a large number of tuples per time unit and “produce” few output tuples
  - Determines scheduling priority of each operator chain
  - Scheduling decisions are made by picking the operator chain with highest priority among those that have operators ready to execute and scheduling the first ready operator in that chain
- Achieves run-time memory minimization
- Suffer from starvation and poor response times during bursts

Discussion

- How does DSMS differ from the standard RDBMS? Can you modify the DBMS to function as a DSMS?
- What could be an efficient way to implement Windowing? Stack, B-Tree, etc.
- What other additions can you make to CQL? Can you add some constraints to reduce latency?

https://tinyurl.com/cse550-dsms

DSMS

How is DSMS different from DBMS:

- Sequential access is necessary; can’t access random records
- Only access limited main memory, DBMS have additional secondary storages

Is it possible to insert Timestamp in RDBMS and use it as a (inefficient) Data Stream Management systems?

Core Difference between DBMS and DSMS

- Type of data
  - DBMS works with “static” data
  - DSMS works with “dynamic” data
- Type of Queries:
  - DBMS: One time queries with a single result
  - DSMS: Continuous Queries, gives multiple results as more tuples arrive. Queries also require a window

Optimizations for DSMS (https://dl.acm.org/doi/abs/10.1145/1379272.1379284)

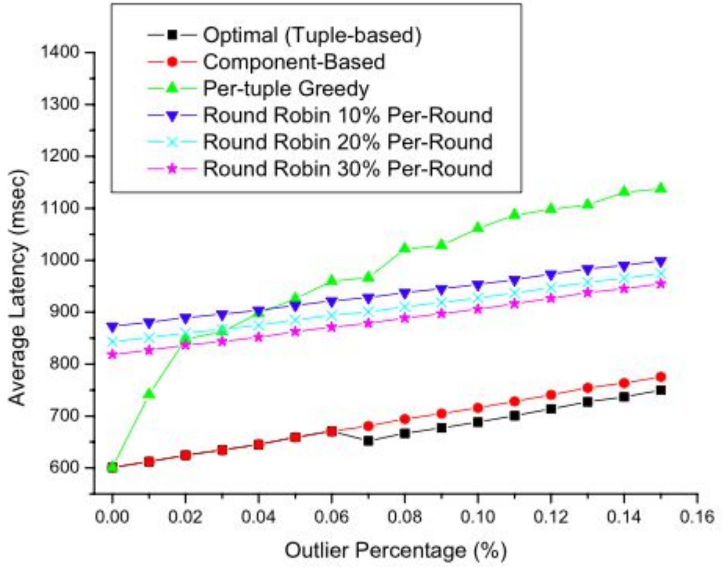


Figure 12: Average Latency by Outlier Percentage

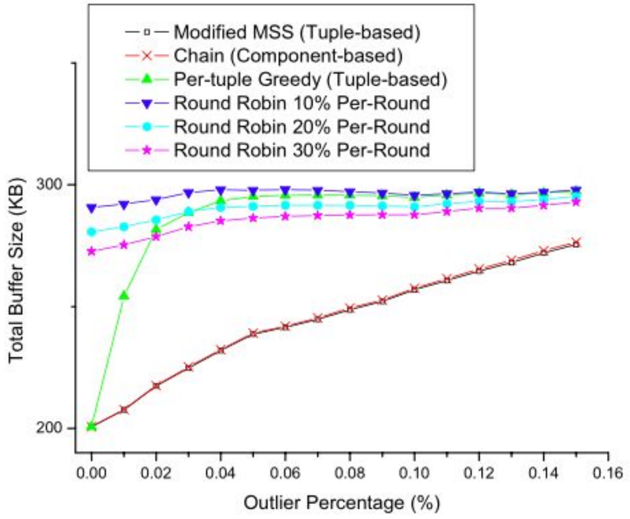


Figure 13: Total Buffer Size by Outlier Percentage

Complex event processing (briefly)

- Similar to how an application can have multiple tables, an application can have multiple streams which may be independent of each other.
- We have to take certain actions based on the state of these streams
- CEP is processing these streams to look for particular patterns/indicators that have certain triggers
- Usually implemented using multiple CQL queries on streams

CEP Uses

- Fraud Detection
- Security Alerts
- Automation
- Event-driven processes

Time Series Database

Time Series:

- Data indexed in time (think of the “versus Time” graph)
- Index is the timestamp for when the data was recorded/measured
- TSDB is an optimization of the general database management system

An example is the InfluxDB

Time Series Data

Influx describes these properties for time series data:

- Billions of individual data points
- High write throughput
- High read throughput
- Large deletes (data expiration)
- Mostly an insert/append workload, very few updates

InfluxDB structure

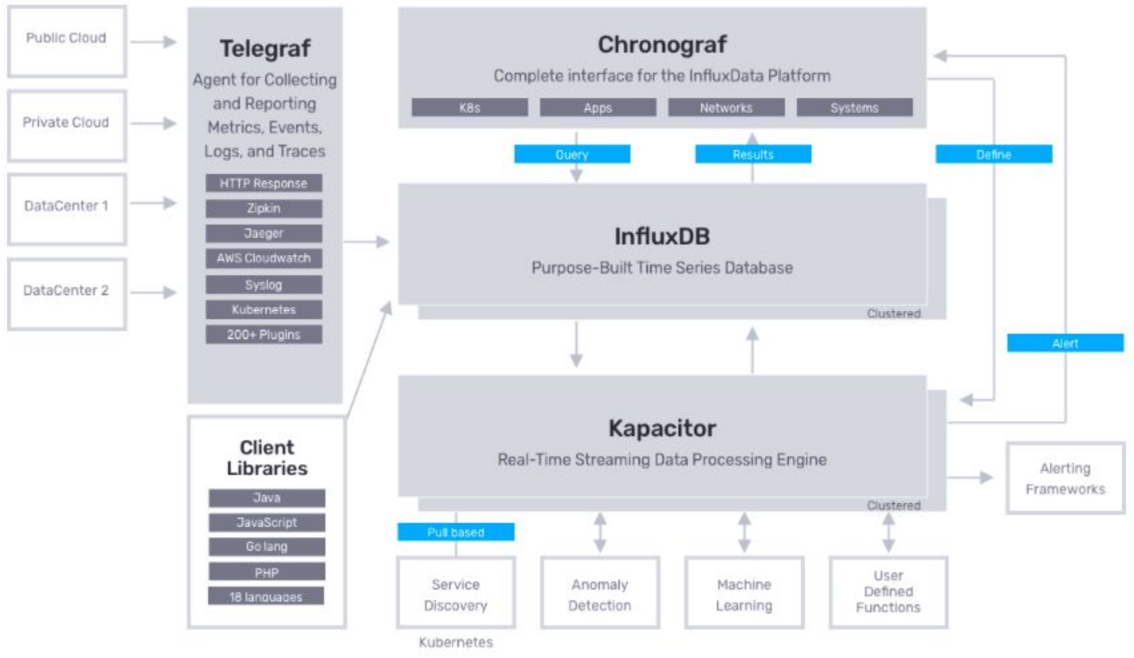
- Write Ahead Logs (WAL) of fixed length
- Cache which the most recent subset of the WAL
- Persistent data

Using in TSDB, after some time period has passed, we delete the individual data points and only store the aggregates.

WAL helps optimize for this high number of deletions

Cache optimizes for quick retrieval

Influx Stream processing



Components of the InfluxData Platform

References

[1] An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations

[2] STREAM: The Stanford Stream Data Manager

[3] https://en.wikipedia.org/wiki/Relational_database

[4] https://dl.acm.org/doi/abs/10.1145/1379272.1379284

[5] https://docs.influxdata.com/influxdb/v1.8/concepts/storage_engine/