Stream Processing

Bounded data - enumeratable/iteratable upon dataset

Unbounded data - only snapshot iteratable; no size property

Batch processing - apply algorithm on bounded dataset; produce single result

Stream processing - algorithm on continuously updating data; continuous results

Cases for stream processing

- intrusion and fraud detection
- algorithmic trading
- process monitoring
- traffic monitoring

Unix stream processing:

- tail/pipe - streaming data acquisition
- pipe - intermediate storage
- applying functions on streaming data
- no windowing (streams into batches)
- no triggers (recomputing when new batch)

Stream processing - techniques and systems that process time stamped events

- component -> acquire event from producer and forward to consumer
- component -> event processor

both components should be scalable, distributable and fault-tolerant

messaging systems - connecting producers to consumers

Unix: tail -f log.txt I wc -l

tail - producer; wc - consumer

pipe - messaging system

- read data from producer and buffer
- block producer when buffer is full
- notify consumer for available data
- publish/subscribe - 1 producer to 1 consumer

publish/subscribe system - connect multiple producers to multiple consumers

direct messaging system - simple network communication (UDP) to broadcast

message brokers/queues - centralised system, reliable message delivery

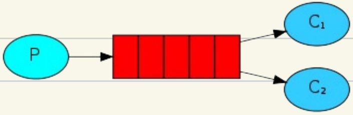
Broker-based messaging

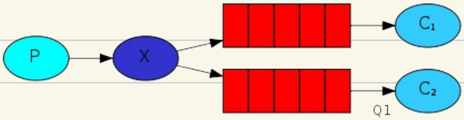
Producer message modes:

- Fire and forget - broke acks message immediately
- Transaction-based - broker writes message to permanent storage before ack

Broker:

- buffer messages, spill to disk
- route messages to queues
- notify consumers



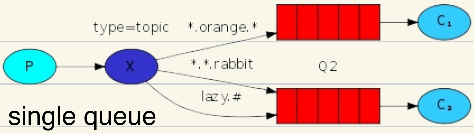


Consumers:

- subscribe to queue
- ack message receipt

Messaging patterns:

- competing worker - multiple consumers on single queue
- fan out - each consumer with replicated queue
- message routing - keys to msg metadata; topic queues specified by key



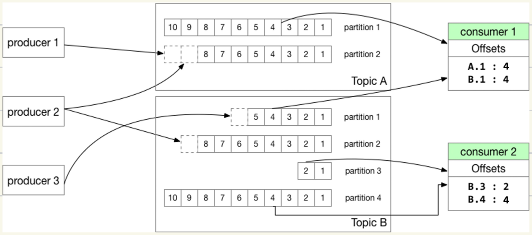
Drawback - after message is received, it disappears

- no message reprocessing
- no proof of message delivery

Log-based messaging

log - append-only data structure ' on disk

- Producer appends message to log
- All consumers connect to log and pull messages
- Broker partitions and distributes log to cluster of machines
- Broker keeps track of message offset for consumer per partition



Programming models

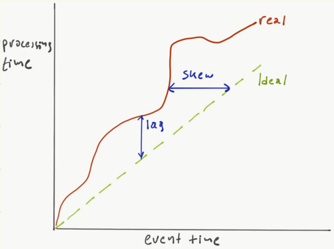
Event sourcing and Command Query Segregation (CQS)

- capture all changes to application state as sequence of events
- event causing mutation on application state in immutable log
- specialised systems for scaling writes and reads, stateless app
- separated continuously updated views of app state
- regenerate application state by reprocessing events

Reactive programming

- declarative programming paradigm - data streams and propagation of change
- event sources as infinite collections, observers subscribe to receive events

Dataflow model



Processing time - time of observation of event in system

Event time - time of event occurrence

t - processing (wall-clock) time

skew = t - s, s - time stamp of latest event processed

lag = t - s, s - actual time stamp of event

4 dimensions of stream processing:

What

- results computed; operations on streams

Element-wise - apply function on individual message

Stream[A].map(x: A -> B): Stream[B]

Stream[A].filter(x: A -> Boolean): Stream[A]

Stream[A].merge(b: Stream[B>:A]): Stream[B]

Stream[A].flatMap(f: A -> Stream[B]): Stream[B]

Stream[A].keyBy(f: A -> K): Stream[(K, Stream[A])]

Stream[A].join(b: Stream[B], kl: A => K, kr: B => K, rs: (A,B) => R): Stream[R]

Aggregations - group events together to apply reduction

Where

- event time computation; streaming windows

Window - static size or time-length batches of data

Session window - dynamically sized, aggregate batches of user activity

When

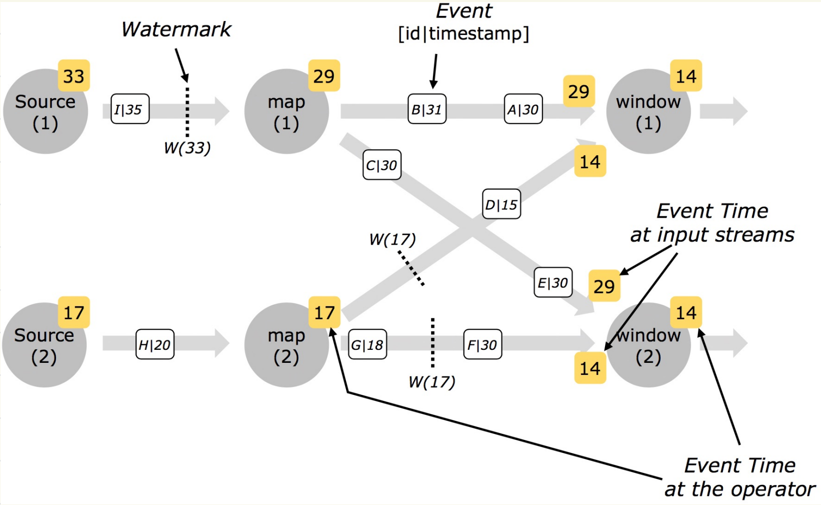
- processing time materialisation/processing

Triggers:

- per-record trigger - fire after a number of records
- aligned delay - fire after a number of seconds across all active windows
- unaligned delay - fire after a number of seconds after first event in window

Watermarks - declaration that all events before this time stamp have arrived

- allow late messages to be processed up to a specific amount of delay



How

- earlier results relation to later refinements

triggers + watermarks = multiple materialisations per window

discard, accumulate or accumulate + retract