11. Stream Processing

- Transmitting Event Streams
  - Messaging Systems
    - Components
      - Event
        - Usually contains timestamp (time of day clock)
      - Producer (publisher or sender)
      - Multiple consumers (subscribers or recipients)
      - Topics (streams)
    - 2 main questions
      - ⚠️ What happens if the producers send messages faster than the consumers can process them?
        - Drops messages
        - Buffer messages in queue
          - What happens when queue grows?
            - System crashes
            - Writes to disk
        - Apply back pressure aka "flow control" (block producer from sending more messages)
      - ⚠️ What happens if nodes crash or temporarily go offline—are any messages lost?
    - Direct messaging from producers to consumers
      - UDP multicast
      - Brokerless messaging libraries
      - StatsD and Brubeck
      - Direct HTTP or RPC request to push messages to the consumer
        - Web hooks
    - Message brokers (message queues)
      - a kind of database that is optimized for handling message streams
    - Message brokers compared to databases
    - Multiple consumers
      - Load balancing
        - Each message is delivered to one of the consumers
      - Fan-out
        - Each message is delivered to all of the consumers
    - Acknowledgments and redelivery
      - Client must explicitly tell the broker when it has finished processing a message
      - After timeout broker redelivers message to another client
        - ⚠️ Such redelivery leads to messages reordering
  - Partitioned Logs
    - Log-based message brokers
      - Hybrid of db and message broker
        - Apache Kafka
        - Amazon Kinesis Streams
        - Twitter's DistributedLog
        - Google Cloud Pub/Sub
        - NATS Steaming
    - Using logs for message storage
    - Logs compared to traditional messaging
      - 👎 When message processing is slow JMS/AMQP is more preferable over Partitioned Log
      - 👍 Where it is fast to process messages log-based approach is
    - Consumer offsets
      - It holds current position in the log
      - If consumer crashes it restarts from the same place
        - ⚠️ But can reprocess last message twice
      - ⚠️ Scaling can be done by keeping offset across few client threads but better create more partitions and use one client = one partition
    - Disk space usage
      - circular buffer or ring buffer but large because it is on disk
    - When consumers cannot keep up with producers
      - Can raise alert
      - If it didn't help and client started missing messages - no problem!
        - Only that client is affected
        - 👍 DEV services can safely consume logs from PROD env!
    - Replaying old messages
      - It is all based on offset which is under client's control
- Databases and Streams
  - Keeping Systems in Sync
    - Batch processes
      - 👎 Can be slow
    - Dual writes
      - 👎 Race condition
      - 👎 One of writes may fail
  - Change Data Capture (CDC)
    - Implementing CDC
      - One db is the leader, others are followers
      - 👍 Log-based message broker is good since it preserves the ordering of messages
      - DB triggers can be used for CDC
        - 👎 But the area fragile and have performance overheads
      - Parsing replication log is more robust
        - 👎 But challenging when schema is changed
        - 👍 There are existing systems that read WAL
          - Can use for audit logs!
          - Bottled Water implements CDC for PostgreSQL using an API that decodes the write-ahead log
          - Maxwell and Debezium do something similar for MySQL by parsing the binlog
          - Mongoriver reads the MongoDB oplog
          - GoldenGate provides similar facilities for Oracle
    - Initial snapshot
      - must correspond to a known position in the change log
    - Log compaction
      - log compaction feature is supported by Apache Kafka
    - API support for change streams
      - Kafka Connect
  - Event Sourcing
    - chronicle data model
    - 🤔 similar to "Command" pattern - Apply/Undo
    - Specialized databases
      - Event Store
    - Deriving current state from the event log
      - 👎 Log compaction is not possible
        - Because inline CDC events don't have all last data
      - Need a mechanism to store snapshot
    - Commands and events
      - Event is successful command
  - State, Streams, and Immutability
    - 👍 The truth is the log
      - The database is a cache of a subset of the log
    - Advantages of immutable events
      - 👍 Useful in finances
      - 👍 Useful for analysis of user's behaviour
    - Deriving several views from the same event log
      - 👍 Running old and new systems side by side
      - 👍 Easier schema to write data when read views are separated
        - Command query responsibility segregation (CQRS)
        - it is entirely reasonable to denormalize data in the read-optimized views
          - Twitter's home timelines are highly denormalised (see Chapter 1)
    - Concurrency control
      - 👎 User can write to the write log, then read from the read view and and not see the change
      - Solutions
        - Sync write to log and update of read views
        - Or use linearisable storage using total order broadcast
      - 👍 From other hand other problems are not present - e.g. events help implementing multi-object transactions
    - Limitations of immutability
      - ⚠️ Sometimes you NEED to rewrite history/physically remove data
        - e.g. GDPR's requirement
- Processing Streams
  - What we can do with the stream
    - Write to storage
    - Push events to the users
    - Process few input streams into few output streams
      - Operator or Job
  - 👍 Uses of Stream Processing
    - Complex event processing (CEP)
      - Searching for specific pattern match
    - Stream analytics
      - Calculating some metrics over events for some time window
      - Sometimes use probabilistic algos that produce approximate results
    - Maintaining materialized views
    - Search on streams
      - Subscribe to search query
    - Message passing and RPC
  - Reasoning About Time
    - Event time versus processing time
      - 👎 Message can be delayed and even reordered - so processing time is sometimes not good
    - Knowing when you're ready
      - ⚠️ It is tricky to understand when time window for event time is closed
        - Can define some wait time
      - 👎 Straggler event can appear after the window was closed and calculated
        - Ignore straggler
          - 👍 And track number of dropped events
        - Publish a correction
      - 👍 Producers can send messages "From now on there will be no more messages with a timestamp earlier than t"
    - Whose clock are you using, anyway?
      - Can log three timestamps
        - 1. Event time
        - 2. Event sent time
        - 3. Event received time
      - 3-2 => offset of time in device and server
        - Assuming network delays are less than required time accuracy
      - 1+offset => actual event time
    - Types of windows
      - Tumbling window
        - Window = 1 minute
          - between 10:03:00 and 10:03:59
          - between 10:04:00 and 10:04:59
      - Hopping window
        - Window = 7 minute; Hop size = 1 minute
          - between 10:03:00 and 10:07:59
          - between 10:04:00 and 10:08:59
      - Sliding window
        - Boundaries are not fixed – they are sliding
      - Session window
        - Session is continued if any event from the same user in 30 minutes
        - In SQL: Group by
  - Stream Joins
    - Stream-stream join (window join)
      - Need to maintain a state
    - Stream-table join (stream enrichment)
    - Table-table join (materialized view maintenance)
      - ❓ Q: how they implement deletes / changes? They should recalculate and delete materialized data
    - Time-dependence of joins
      - Slowly changing dimension (SCD)
        - 👍 Better to refer the unique version of item that is slowly changing
  - Fault Tolerance
    - In batch processing - exactly-once semantics
    - Microbatching and checkpointing
      - Typically microbatch is around one second
      - 👎 But side-effects can exist - e.g. delicate sent emails
    - Atomic commit revisited
    - Idempotence
      - goal is to discard the partial output of any failed tasks
        - One approach - Distributed transaction
        - 👍 Another approach - Idempotence
      - 👍 Idempotent operation - operation that you can perform multiple times - and it has effect as if you performed it only once
      - A bit of extra data usually helps!
        - 👍 e.g. adding log's offset to database update record
    - Rebuilding state after a failure
- Summary
  - Similar to batch processing but data is unbounded
  - Transportations
    - Direct messaging
    - Message brokers
    - Event logs
  - Source of streams
    - User activity
    - Sensors
    - Data feeds
    - Writes to DB
  - 3 types of joins
    - Stream-stream joins
    - Stream-table joins
    - Table-table joins