Streaming Systems
Streaming 101

Chapter 1

1.1

- Motivation:
  - Timely
  - Unbounded dataset [what the author wants to stress, instead of latency/approximate]
  - Workload evenly over time
- Dataset shape:
  - Cardinality
    - Bounded / unbounded
  - Constitution
    - Table /Stream

1.1

- Streaming can & cannot do:
  - Lambda Architecture:
    - Low-latency, inaccurate
    - along side a batch system, and bs rolls along (merge results at last)
    - Maintaining a Lambda is a hassle
  - Goal: strongly consistent streaming engine
    - Repeatability + replayable (like Kafka)
    - A strict superset of batch functionality (users choose an appropriate efficiency level)
  - How to beat batch:
    - Correctness (parity with batch)
    - Reasoning about time (beyond batch)

1.1

- Correctness [1]
  - Boils down to consistent storage. Need a method for checkpointing persistent state over time (think about spark streaming). (but many systems try to get by without strong consistency. Maybe inaccurate. Pick up them carefully based on the demand)
  - Strong consistency is required by exactly-once processing.
    - MillWheel, Spark streaming, Flink snapshotting
- Reasoning about time [2]
  - Unbounded, unordered data of varying event-time skew

1.2

- Event time / Processing time
  - ET: characterizing user behavior over time, most billing applications, and many types of anomaly detection
  - skew/lag: variable function of the characteristics of the underlying input sources, execution engine, and hardware. [Real distributed scenes: hard]
    - Shared resource limitations, like network congestion, network partitions, or shared CPU in a nondedicated environment
    - Software causes such as distributed system logic, contention, and so on
    - Features of the data themselves, like key distribution, variance in throughput, or variance in disorder (i.e., a plane full of people taking their phones out of airplane mode after having used them offline for the entire flight)

1.2

- Windowing by processing time
  - Impact correctness if your algorithm/use cases need to keep the event time order. Otherwise good like calculate rates of web requests
  - Say, some approximation algorithms / lstm needs the input to be well-ordered
- Windowing by event time
  - Do not have a mapping between processing time and event time. Thus needs longer time for **buffering**. Temporal shuffle is needed to put data into their correct locations
  - Now supported naively by Spark, storm, apex.
  - unbounded-> not complete (completeness)
- Completeness:
  - Say, given window [t1, t2], we do not know if and when we've seen all data for given window
  - Reasonably accurate heuristic estimate of window completion: watermarks

1.2

- Bounded Data: ~
- Unbounded Data: Batch
  - Fixed windows(hard to keep completeness due to delay or mitigation) /sliding windows /sessions (dynamic windows)
- Unbounded Data: Streaming
  - Highly unordered with respect to event times,
  - Of varying event-time skew, meaning that you can't just assume you'll always see most of the data for a given event time X within some constant epsilon of time Y
  - Approaches:
  - Time-agnostic (Filtering/inner join) Not for outer join(has a timeout for completeness) ;
  - Approximation algorithms (top-n, k-means): has some predicate: data arriving in event order!
  - Window: event time/processing time