CSE 550 11/30/22: STREAM

Alexandra Michael and Mengqi Chen

STREAM:
The STanford StREam DatA Manager

Background: Data Stream Management Systems (DSMS)

- AKA "continuous query systems" or similar
- Distinguished from DBMS's (database management systems) by **continuous, high-volume data streams**

Meaning…

- Time-varying, potentially infinite datasets
- Highly limited memory relative to data quantity, with no random access
- Long-running queries over current and incoming data

STREAM (2003) one of several DSMS or related projects in the early 2000's (Berkeley's TelegraphCQ ('03); University of Wisconsin-Madison's NiagaraCQ ('00); others)

STREAM Overview

- Input Streams
- Scratch Store
  - = intermediate state
- Archive
  - storage for preservation or offline processing
- Continuous Queries
  - user/application query, active until deregistered
- Results
  - streamed, or stored and updated over time

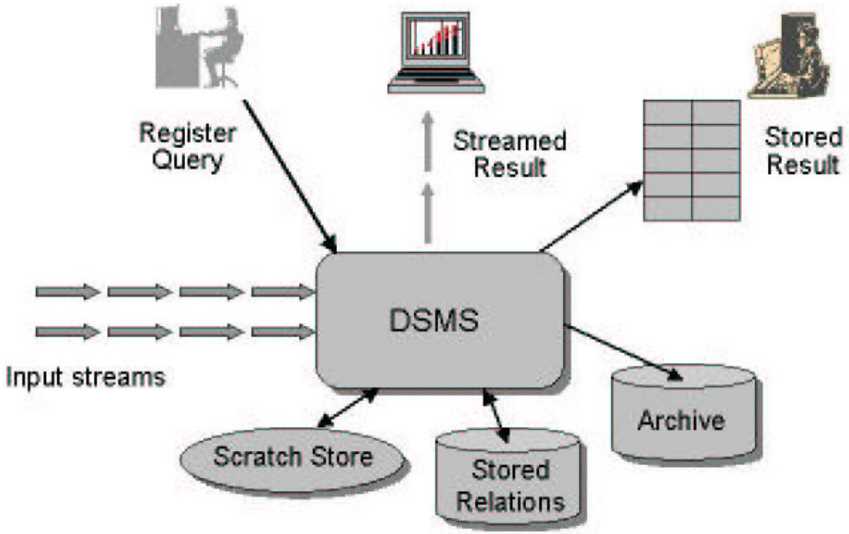


Figure 1: Overview of STREAM

CQL (Continuous Query Language)

- a ***stream*** is a bag of *(tuple, timestamp)* pairs
  - unbounded, append-only
- a ***relation*** (i.e., over data) is a time-varying bag of *tuples*
  - supports update, insert, delete
- *streams* mapped to *relations* via ***windowing*** operators
  - Once mapped, relations are transformed and re-streamed as output

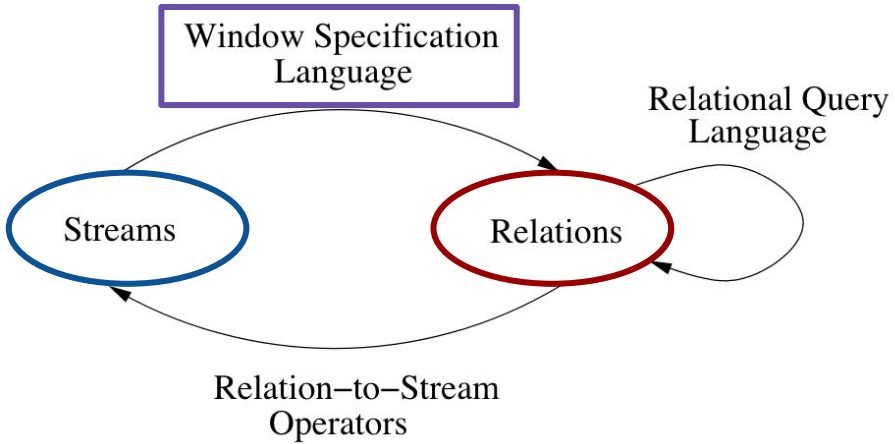


Figure 2: Mappings used in abstract semantics

STREAM Query Processing

1. Query specified in CQL
2. Register query with STREAM
3. Query compiled into *query plan*, including:
   a. *Query operators* that read, process, and write tuples as output
   b. Q*ueues* to buffer operator input (either from input stream or another operator's output)
   c. *Synopses* to maintain operator runtime state
4. Query plan is merged with existing ones where possible

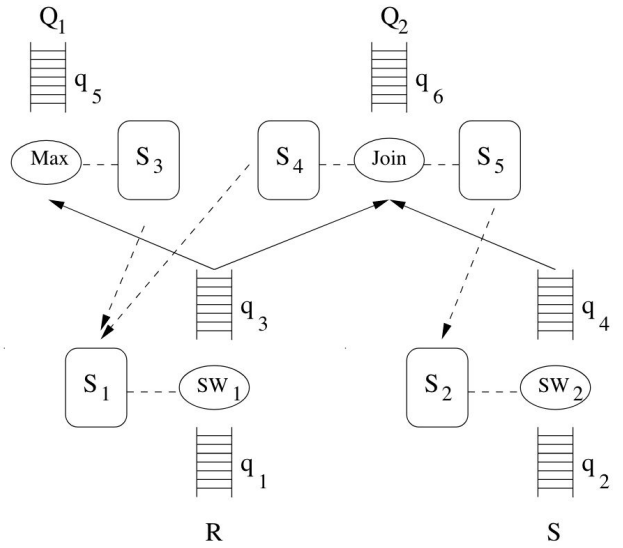


Figure 3: STREAM query plans

STREAM Operator Scheduling

Goal: minimize memory required for backlog buffering

Solution: *Chain scheduling*

- Breaks up query plans into disjoint chains of consecutive operators
  - E.g., Op1 → Op2 → Op3 could be broken into Op1 → Op2 and Op3
- Bases decisions on which operators consume input most quickly and produce least, slowest output
- Schedules the operator chain with highest priority among those that are ready
- "Near-optimal" in some cases, performs well in others
- May suffer starvation, poor response times during input bursts

Ongoing Research

- Efficient query processing
  - Techniques for sharing computation and memory resources among plans, and more
- Cost-based optimization and resource allocation (for query plan generation)
- Scheduling
  - Optimizing chain scheduling to reduce latency during bursts
- Graceful degradation under overload
  - E.g., by *load shedding* to selectively drop excess input tuples
- Distributed stream processing

Discretized Streams: Fault Tolerant Streaming Computation at Scale

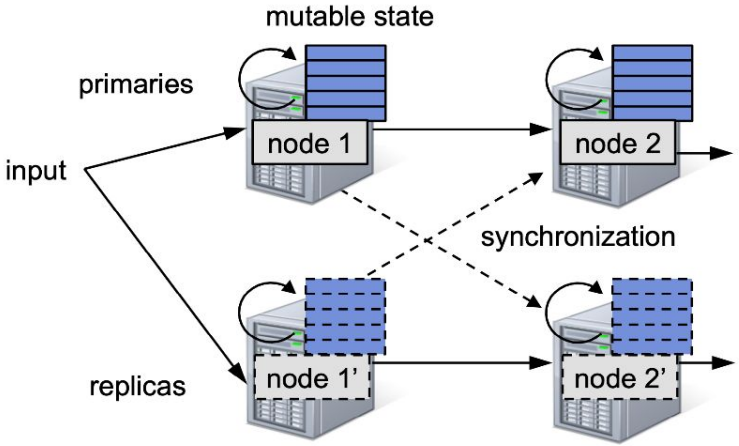
Data Stream Computation at Scale

- Many Big Data applications need to handle large amounts of data in real time
- Many distributed stream processing systems do not provide efficient fault recovery and do not handle straggler nodes
- This is because of the usage of a *continuous operator* model, where streaming computations are done via long-lived stateful operators
- Replication or upstream backup is needed for fault tolerance
- Node startup, state copy, and node overtake is needed for straggler nodes

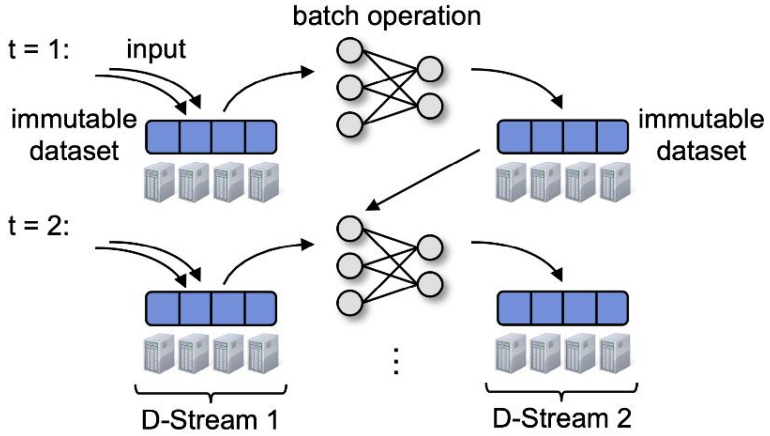
D-Streams

- Discretized Streams (D-Streams) solve the fault recovery and straggler issues by having *short, stateless, deterministic* tasks.
- **Short**: Divide stream data into batches
- **Stateless**: No need for internal state for example to keep track of computation progress. Instead, store state through *Resilient Distributed Datasets* (RDD)
- **Deterministic**: each RDD can be recomputed deterministically by using lineage information
- A D-Stream is a sequence of partitioned RDDs that can be acted on by transformations (e.g. map, reduce)

Record-at-a-Time versus D-Streams



(a) Continuous operator processing model. Each node continuously receives records, updates internal state, and emits new records. Fault tolerance is typically achieved through replication, using a synchronization protocol like Flux or DPC [34, 5] to ensure that replicas of each node see records in the same order (*e.g.*, when they have multiple parent nodes).



(b) D-Stream processing model. In each time interval, the records that arrive are stored reliably across the cluster to form an immutable, partitioned dataset. This is then processed via deterministic parallel operations to compute other distributed datasets that represent program output or state to pass to the next interval. Each series of datasets forms one D-Stream.

Example

- Example of a program that computes a running count of view events by URL by using the Apache Spark API

```
pageViews = readStream("http://...", "1s")
ones = pageViews.map(event => (event.url, 1))
counts = ones.runningReduce((a, b) => a + b)
```

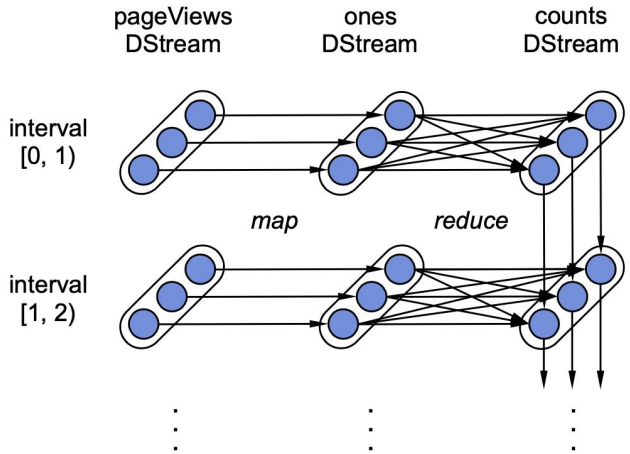


Figure 3: Lineage graph for RDDs in the view count program. Each oval is an RDD, with partitions shown as circles. Each sequence of RDDs is a D-Stream.

D-Streams in Spark Streaming System

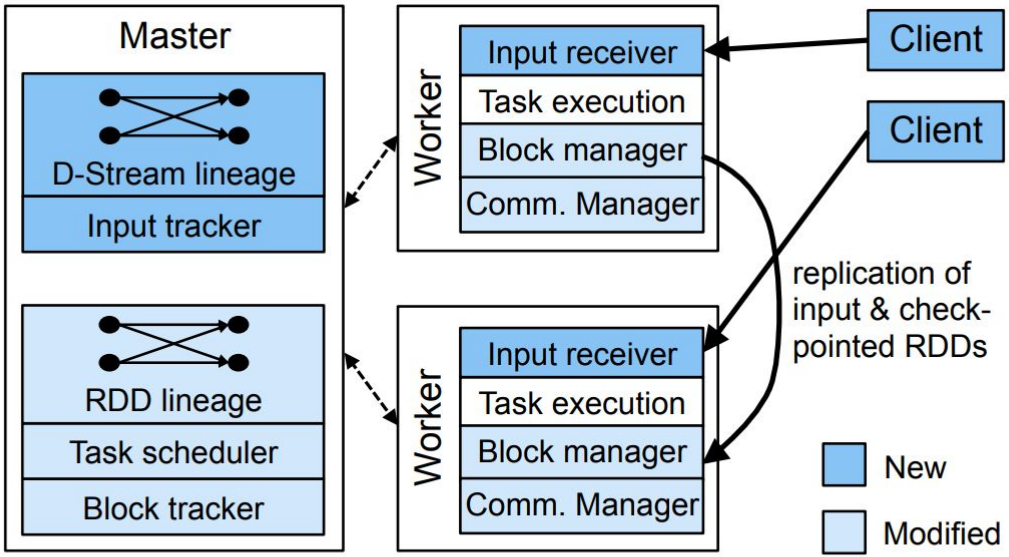


Figure 6: Components of Spark Streaming, showing what we added and modified over Spark.

Apache Flink: Stream and Batch Processing in a Single Engine

Data Processing at Scale

- Data processing can be done via Data-stream processing or batch data processing. These two were considered as two different types of applications.
- Continuous amounts of data arrive, and some application may ignore the continuous data flow nature, and instead divide the data into batches
- Flink proposes a unified architecture that can process both data streams and data batches via its distributed dataflow engine

Flink Software Stack

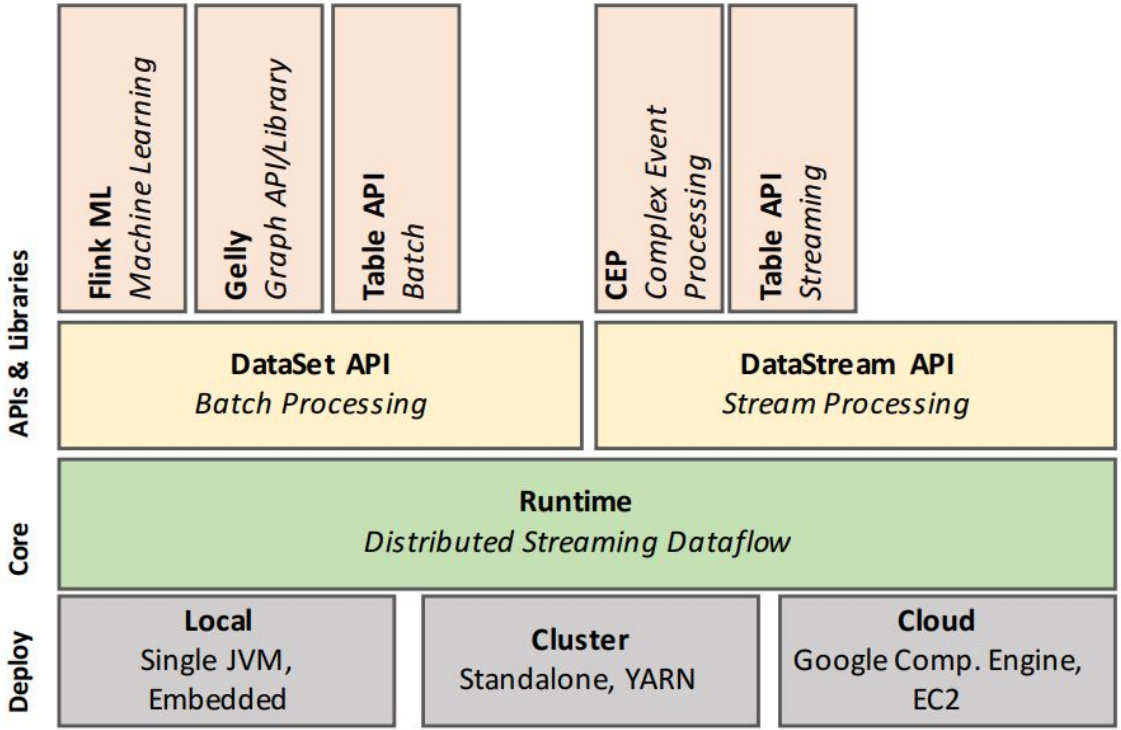


Figure 1: The Flink software stack.

Flink Process Model

- Three types of process in a Flink cluster
- Client takes program code, transforms to dataflow graph and submits to job manager
- Job Manager coordinates distributed execution
- Task Manager does actual data processing

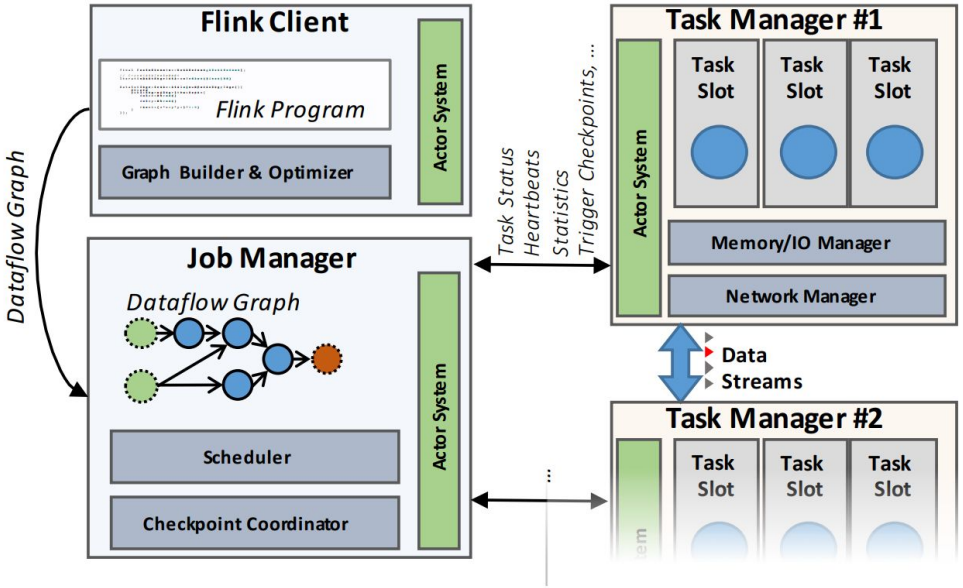


Figure 2: The Flink process model.

Flink Dataflow Graph

- Stateful operators that implement all of the processing logic (e.g. filters, hash joins, stream window)
- Data streams produced by an operator are available for consumption by another operator
- Pipelined and Blocking data exchanged
- Control Events (checkpoint barriers, watermarks, etc.)
- Used for both stream and batch processing

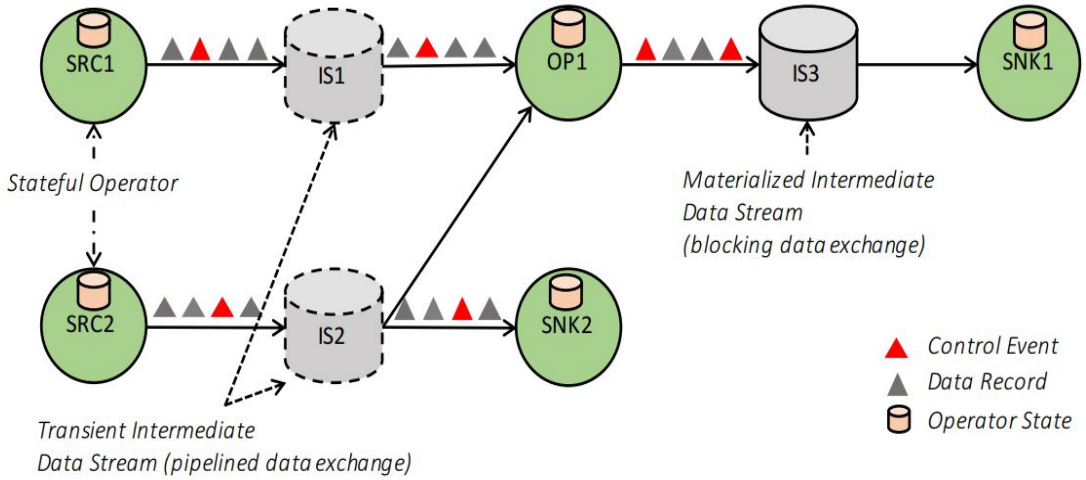


Figure 3: A simple dataflow graph.

Fault Tolerance

- Achieved through Asynchronous Barrier Snapshotting (ABS)
- Barriers are control records injected into the data streams with a logical time to separate a stream into parts
- An operator performs alignment phase when receiving barrier, and write state to durable storage

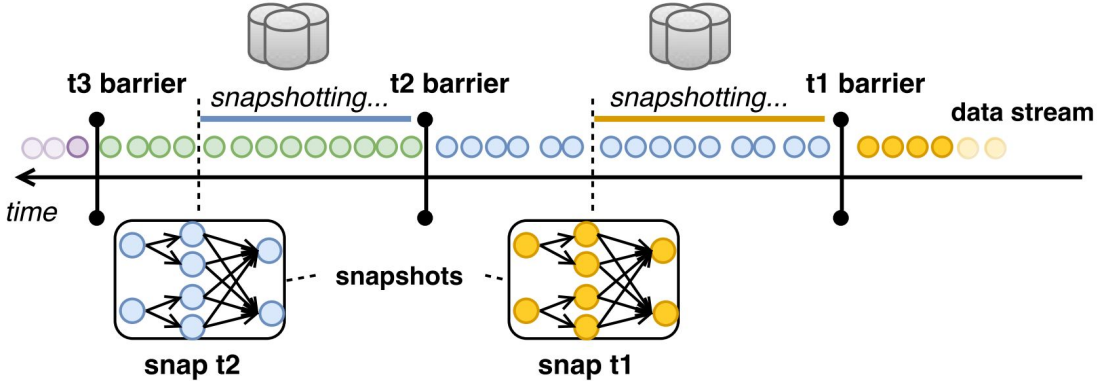


Figure 5: Asynchronous Barrier Snapshotting.

Kafka: a Distributed Messaging System for Log Processing

Distributed Log Processing at Scale

- Use of log data in real-time features requires processing vastly higher volumes than for previous uses
- Existing systems primarily scrape and store log data for offline use; insufficient for use in online, real-time feature production
- Kafka: LinkedIn's distributed messaging system for log processing, allowing real-time consumption of log events by applications

Kafka Architecture

- Producers publish messages to *topics*
- Published messages stored on *brokers* (servers)
- Consumers subscribe to topics from brokers and pull messages from the subscribed topics

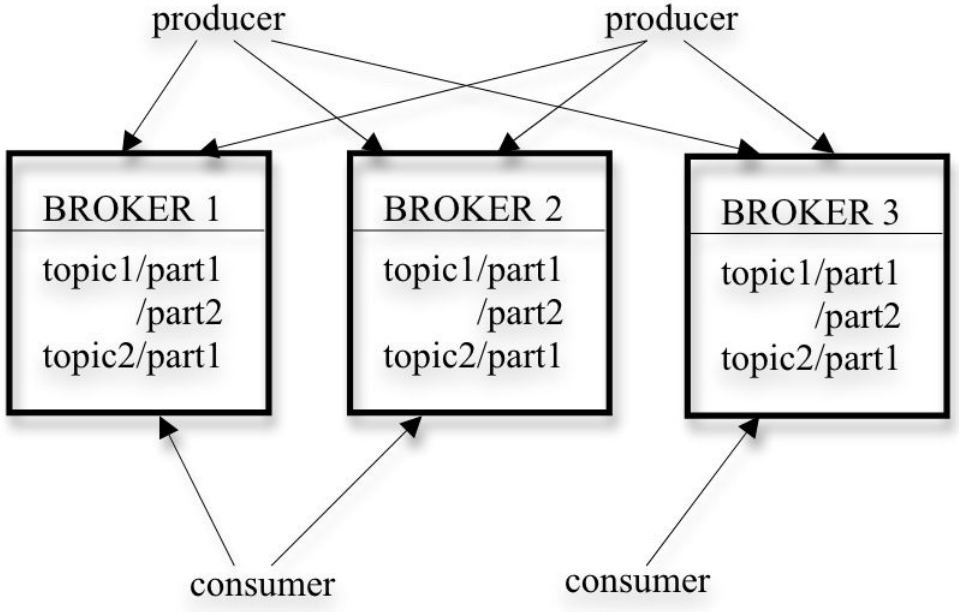


Figure 1. Kafka Architecture

Kafka Design Principles

- Efficient storage and data transfer
  - Simple storage mechanism (one topic partition = one 1G logical log)
  - Efficient data transfer with minimum transfer sizes, stateless brokers
- Distributed coordination
  - Consumers grouped to jointly consume messages by topic (evenly divides message load among consumer group)
- Delivery: only guarantees at-least-once delivery

Experimental Performance

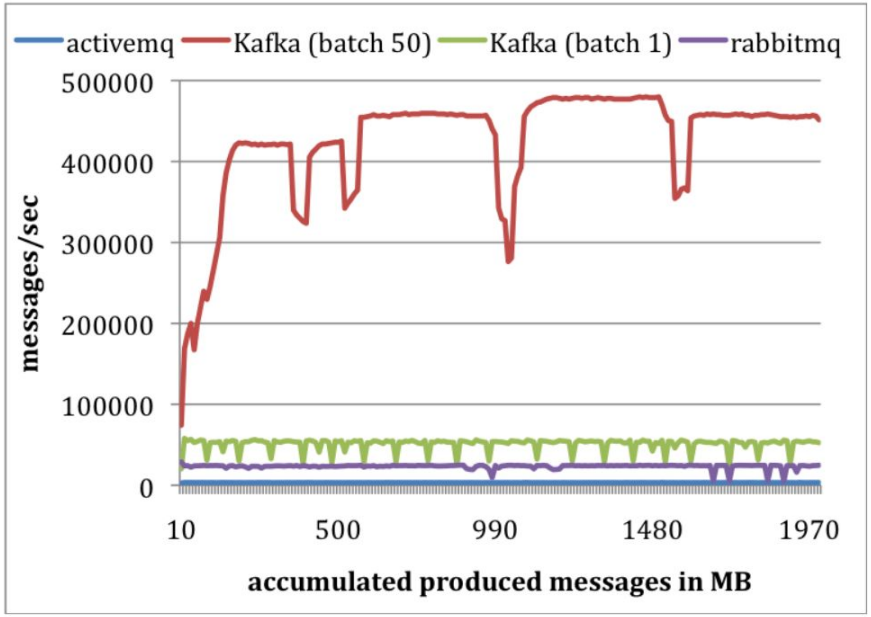


Figure 4. Producer Performance

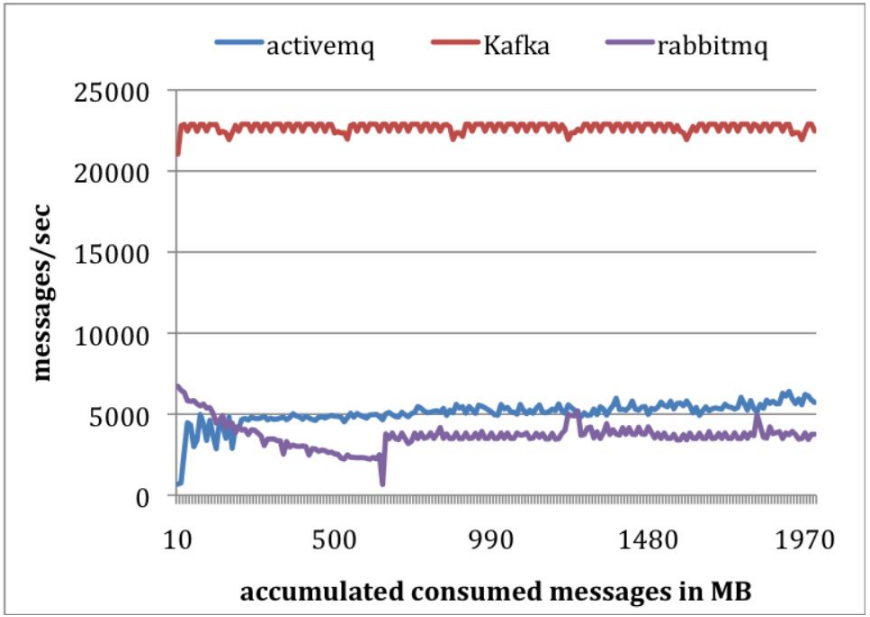


Figure 5. Consumer Performance

Discussion Q1

- Describe a query optimization strategy of your choice/Discuss how to address a shortcoming of chain scheduling.

Discussion Q2

- Based on the various systems we've considered, how will you propose a sample distributed stream processing system (as mentioned in the future works)

- Distribute operations at different nodes (e.g. pipelines)
- Distribute a single operation across different nodes, need a node to combine results
- Challenge of having global time that serializes the data stream
  - Logical Clocks
  - Spanner's TrueTime
  - Marzullo's algorithm

Discussion Q3

- Compare one similar and one different characteristic between DBMS and DSMS

Similarities
- Relational view of data
- Language
- Query processor
- Operator scheduling
- Optimizations (filtering pushdown, join optimization)

Differences
- Types of query (one-time vs continuous)
- Memory challenges
- Time requirements (timestamps, real-time requirements for DSMS)
- Single query latency vs average latency