Babeș-Bolyai University

BACKPRESSURE MECHANISMS IN STREAMING SYSTEMS KAFKA VS FLINK

Author: Turcu Ciprian-Stelian

THE BACKPRESSURE PROBLEM

What is Backpressure?

- Flow control mechanism when producers send data faster than consumers can process
- Without backpressure: system crashes, data loss, or memory overflow
- Critical for system stability under variable loads

Why It Matters in Big Data

- Billions of events per day in production systems
- Processing rates vary by operator complexity
- Temporary load spikes are inevitable
- Must maintain data integrity and low latency

Real-World Scenario

- Kafka topic receiving 1M events/sec
- Consumer can only process 500K/sec
- What happens to the other 500K events?

TWO DIFFERENT PHILOSOPHIES

1

Pull-Based Flow Control (Kafka)

- Consumer explicitly requests data at its own pace
- "I'll take records when I'm ready"
- No upstream feedback to producers
- Broker acts as durable buffer

Push-Based Backpressure (Flink)

- Framework automatically throttles data flow
- "Stop sending, my buffers are full"
- Backpressure propagates upstream automatically
- Built into network layer

2

KAFKA'S PULL-BASED APPROACH

How It Works

1. Consumer calls poll() to fetch records from broker
2. Consumer controls batch size with max.poll.records
3. Kafka topics store data durably on disk
4. Pull frequency determines consumption rate

Key Configuration Parameters

- max.poll.records: Records per poll
- max.poll.interval.ms: Max time between polls
- fetch.min.bytes: Minimum data before broker responds
- enable.auto.commit: Automatic vs manual offset commits

FLINK'S CREDIT-BASED BACKPRESSURE

How It Works

1. Credit-based flow control (similar to TCP sliding window)
2. Automatic propagation through entire pipeline
3. No application code needed

The Credit System

- Downstream announces: "I have N buffers available"
- Upstream sends backlog: "I want to send M messages"
- Downstream grants credits: "Send X buffers"
- Upstream sends only X buffers
- Each buffer sent = -1 credit

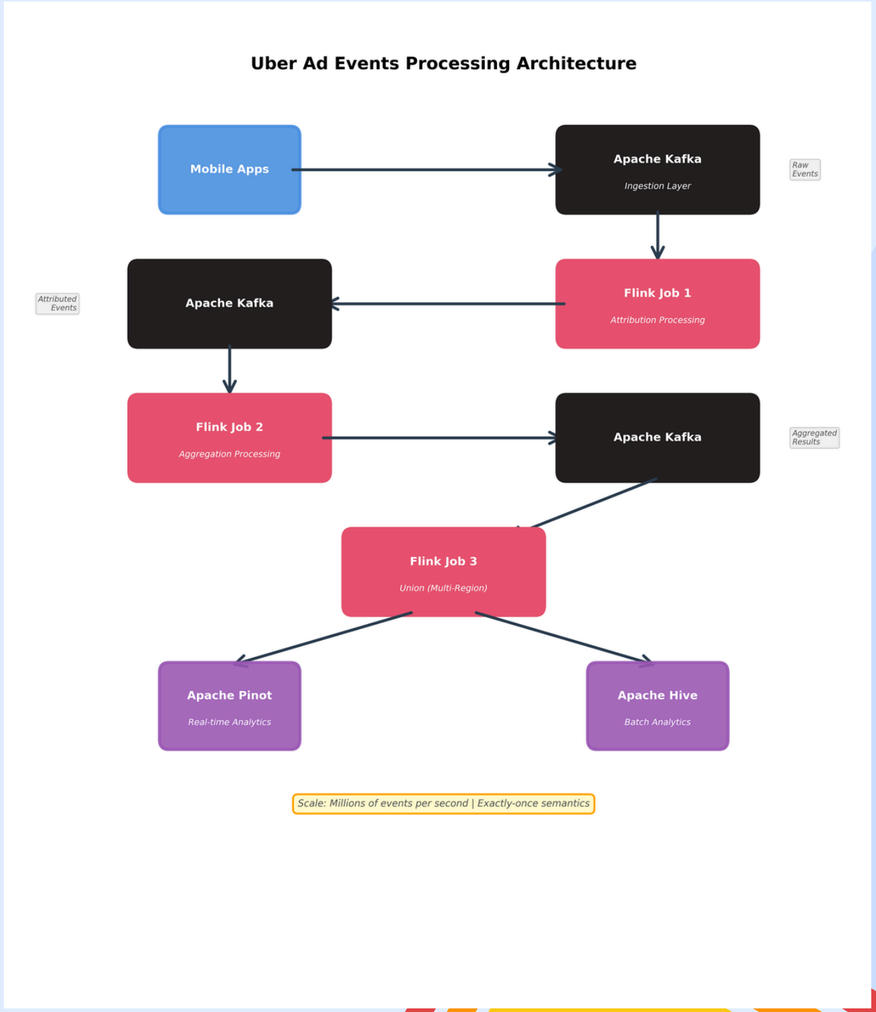
UBER AD EVENTS

System Requirements

- Process millions of ad events per second
- Exactly-once semantics end-to-end
- Sub-second latency for real-time decisions
- Multi-region deployment for resilience

Why Both Technologies

- Kafka: Durable buffering, handles producer spikes
- Flink: Complex stateful aggregations, exactly-once processing
- Kafka again: Decouples processing stages
- Final sinks: Pinot (real-time) + Hive (batch analytics)



NETFLIX RDG

Real-Time Distributed Graph

- Track member interactions across streaming, live events, games
- ~1 million messages/second per Kafka topic
- Near-real-time graph updates for personalization

Technology Choice

- Kafka: Event ingestion from Netflix apps
- Flink: "Strong capabilities around near-real-time event processing"
- Flink: Filtering, enrichment, graph transformations
- Each job independently tuned for its topic's characteristics

Architecture Evolution

- Initially: One Flink job consuming all Kafka topics
- Problem: "Became operational nightmare, tuning extremely difficult"
- Solution: 1-to-1 mapping of Kafka topic to Flink job
- Trade-off: More jobs but simpler operations

ADVANTAGES & TRADE-OFFS

Kafka Advantages

- Simple pull model, easy to understand
- Durable buffering survives failures
- Complete producer/consumer decoupling
- Language-agnostic consumers
- Mature operational tooling

Kafka Disadvantages

- Manual flow control implementation required
- Rebalancing storms from slow consumers
- max.poll.interval.ms false failures
- Consumer lag can grow unbounded
- Downstream systems can't signal back

Flink Advantages

- Automatic backpressure, no code needed
- True streaming, low latency
- Exactly-once guarantees
- Rich operators (windows, joins, aggregations)
- Built-in monitoring dashboard

Flink Disadvantages

- Operational complexity (cluster management)
- Memory-intensive for large state
- Steep learning curve
- Requires external durable source (Kafka)
- More components to manage

MONITORING IN PRODUCTION

Kafka Critical Metrics:

- consumer-lag: Offset difference
- records-consumed-rate: Throughput
- rebalance-count: Stability indicator
- fetch-latency-avg: Broker performance

Flink Critical Metrics:

- backPressuredTimeMsPerSecond: Time under backpressure
- idleTimeMsPerSecond: Waiting for data
- busyTimeMsPerSecond: Actual work time
- inPoolUsage/outPoolUsage: Buffer utilization

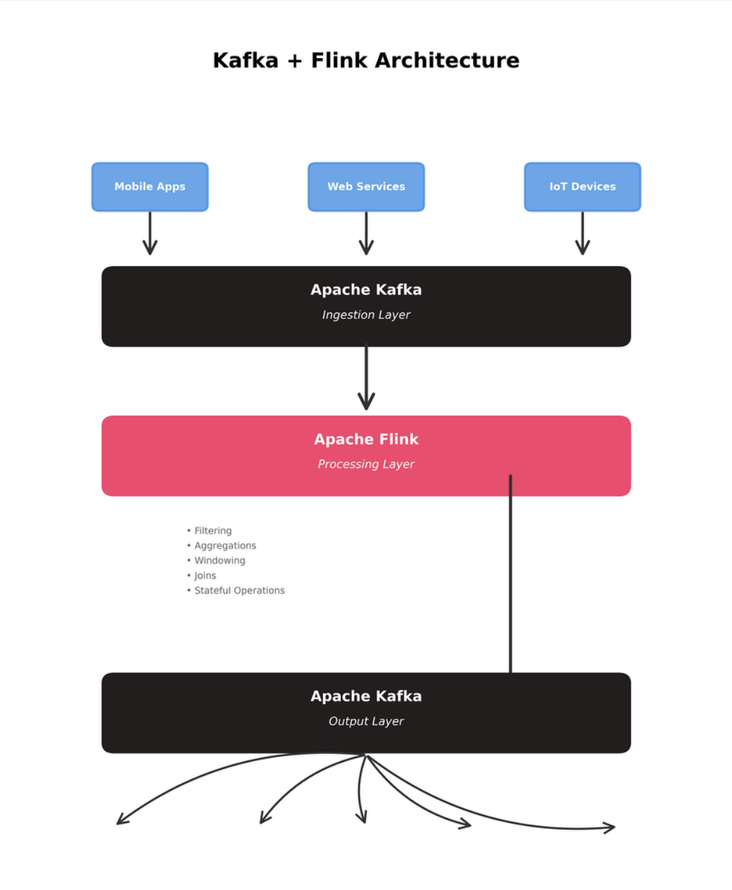
WHY USE BOTH TOGETHER

Kafka excels at:

- Durable message storage
- Decoupled communication
- Multi-consumer scenarios
- Inter-service messaging

Flink excels at:

- Complex stateful processing
- Low-latency transformations
- Windowing and temporal joins
- Exactly-once computations



Production Pattern:

- Kafka: Separation of concerns
- Flink: Complex transformations
- Kafka: Result distribution
- Both: Independent scaling

CONCLUSIONS & RECOMMENDATIONS

When to Choose Kafka Pull-Based

- Simple event-driven microservices
- Variable consumer processing rates
- Need maximum flexibility and decoupling
- Durability more important than millisecond latency

When to Choose Flink Credit-Based

- Complex stateful transformations required
- Sub-second latency critical
- Windowing, joins, aggregations needed
- Want automatic backpressure handling

When to Use Both

- Production Big Data platforms
- Need durability AND complex processing
- Kappa architecture (streaming-only)
- Scale requires separation of concerns

Babeș-Bolyai University

THANK YOU

Turcu Ciprian-Stelian | 2026