TRACK
Modern Data Architectures, Pipelines, & Streams

SESSION

Microservices to Async Processing Migration at Scale

Sharma Podila
Software Engineer @Netflix

Sharma Podila shares from their experience migrating to asynchronous processing at scale, requiring attention to managing data loss, a highly available infrastructure, and elasticity to handle bursts.

Structure of this talk

- Motivations
- System design details
- Challenges, choices, trade offs
- Validation and rollout
- Summary of results

Netflix streaming for over 200 Million members

Netflix streaming for over 200 Million members

Collect operational and analytical data during playback

Product features: viewing history, “continue watching”

Feed personalization and recommendations engines

Business analytics

Netflix streaming for over 200 Million members

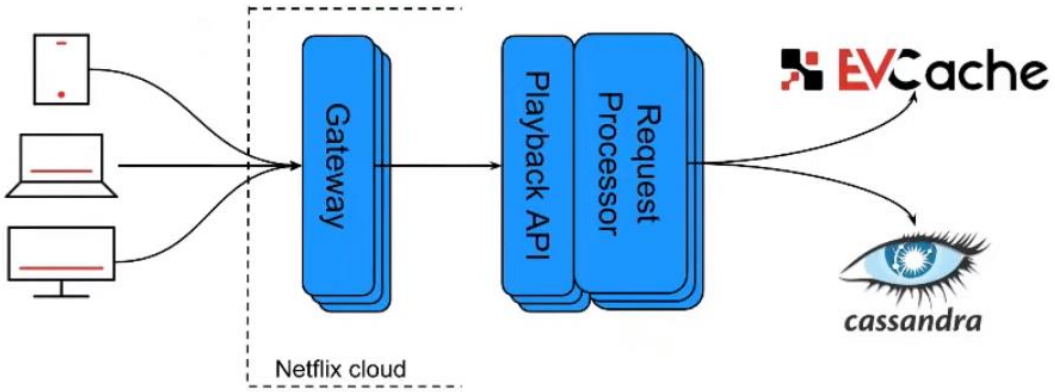
Product feature, Viewing history

Members can see their viewing activity or optionally hide it

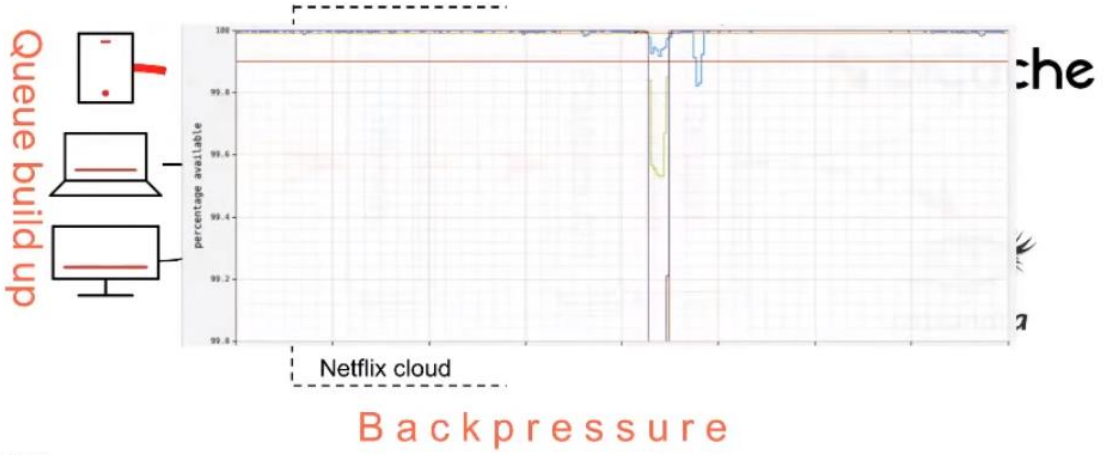
Low latency global materialized views

Motivation

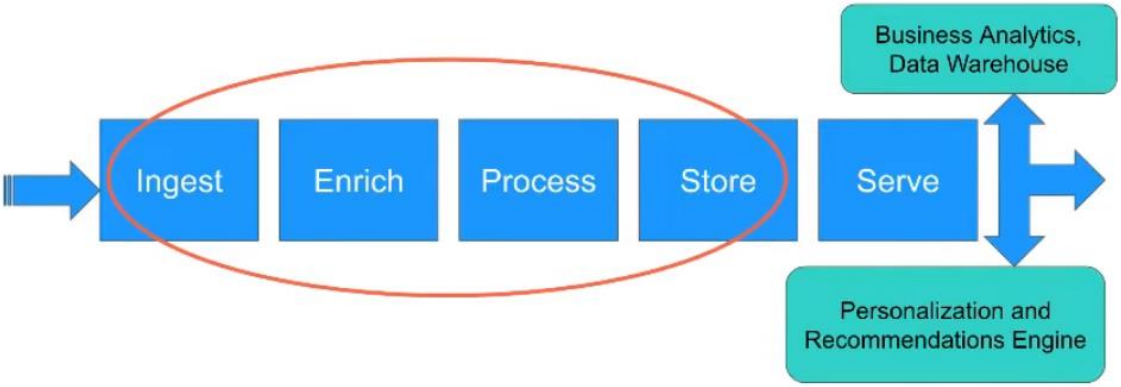
Existing architecture



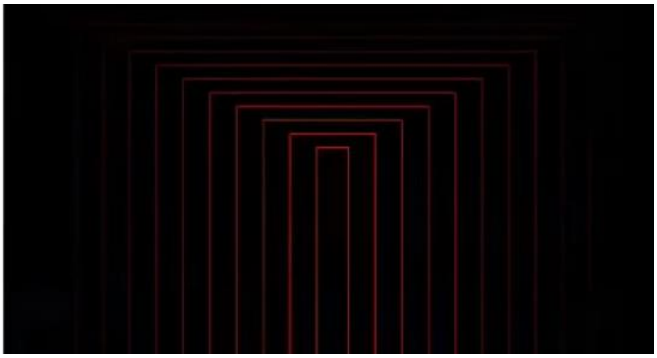
Existing architecture



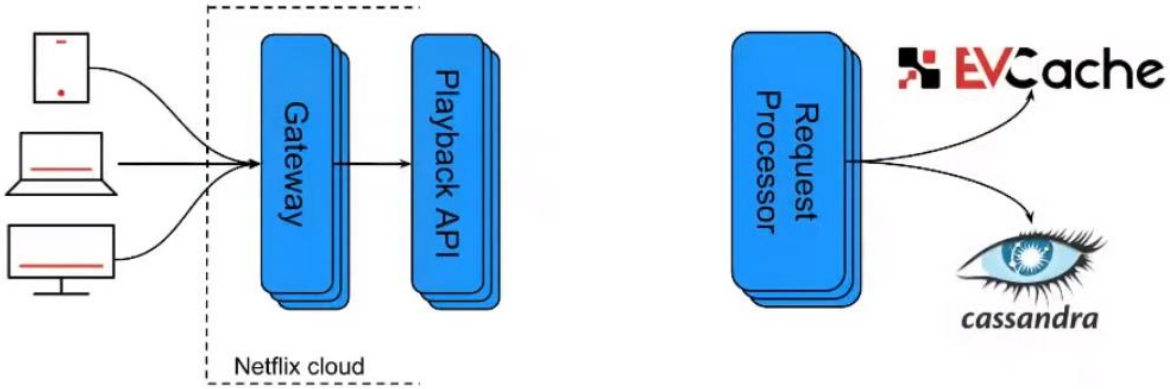
Data Processing Stages



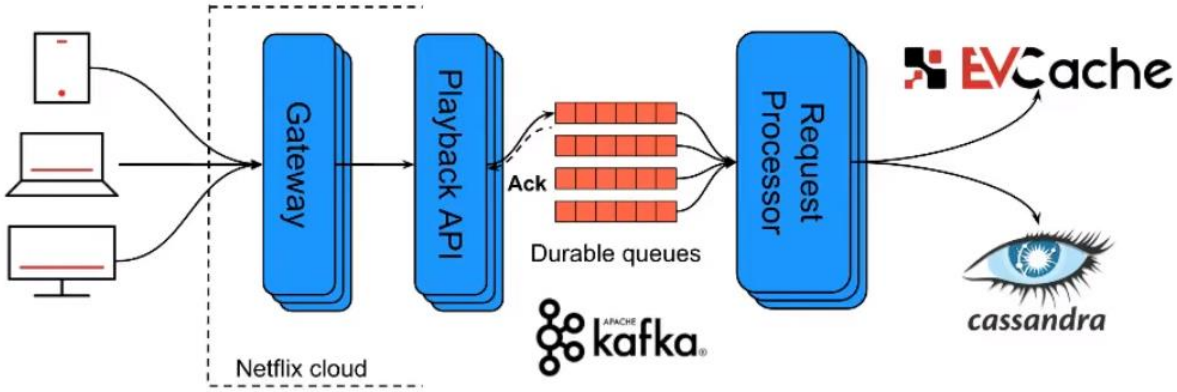
System design changes



New async architecture



New async architecture



This is on a scale of 1million/messages per second

Challenges in Async processing

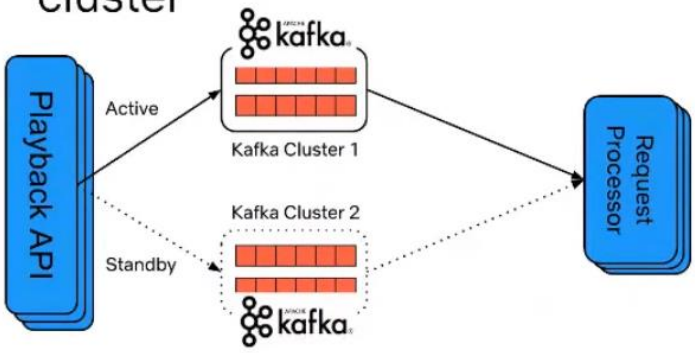
- Data loss
- Processing latencies
- Out of order and duplicate records
- Consumer platform choice
- Intermittent processing failures
- Cross region routing

Challenges in Async processing

Data loss

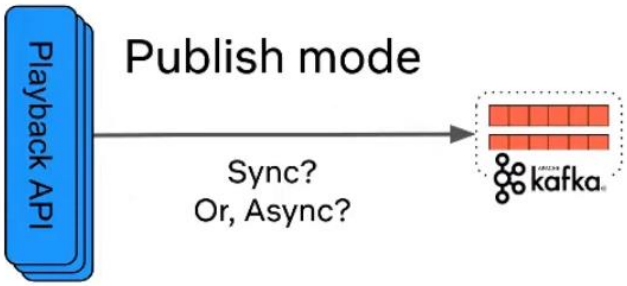
— Kafka cluster unavailability (rare)

— Improve availability with a standby cluster



Challenges in Async processing

Data loss



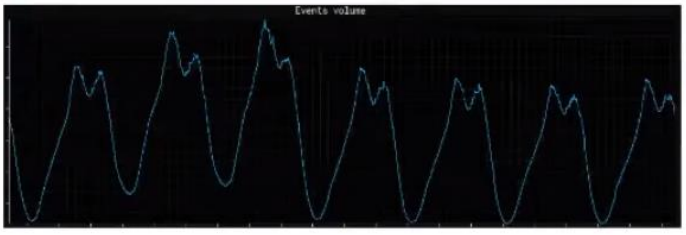
- "ack=all" vs "ack=1" for publish
- Potential data loss from "unclean" broker leader election
- Producer library optimizations
  - Pick another partition if using non-keyed partitions
  - Avoid under-replicated partitions
- Async publish mode for scale and monitor data durability

Challenges in Async processing

— Inherent latency with Kafka is low

Processing latencies

— How to handle lag from traffic surge?



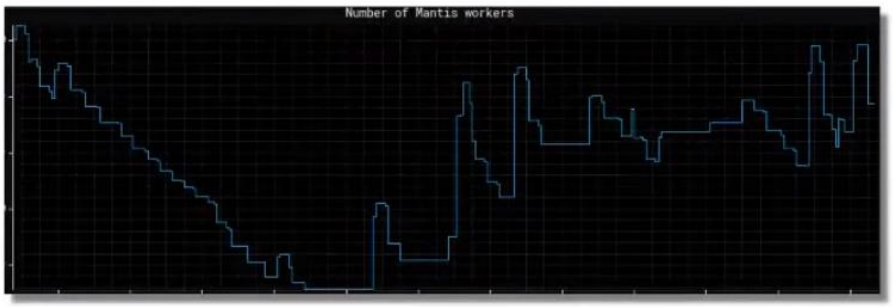
— Provision for peak vs. autoscale

Challenges in Async processing

Processing latencies

— Changes to number of consumers:
- Resource efficiency vs partitions rebalance

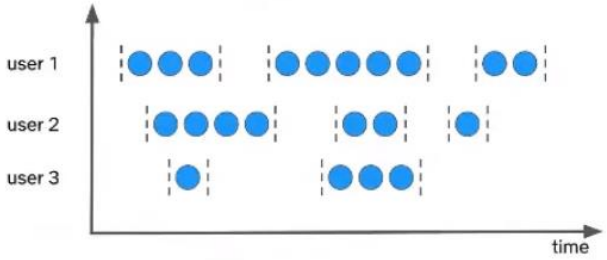
— What metric to use for autoscaling?
- Lag helps scale up but not scale down
- CPU or *rps* works in practice



Challenges in Async processing

Out of order and duplicate records

— Windowing or sessionization



— Leverage application specifics
E.g., associate by a session id

— Deduplication
- E.g., server timestamp to resolve writes

Challenges in Async processing

What's the best consumer platform?

~~Which processing platform should I use?~~
Which processing platforms benefit which use cases?

https://github.com/Netflix/mantis

Apache Flink

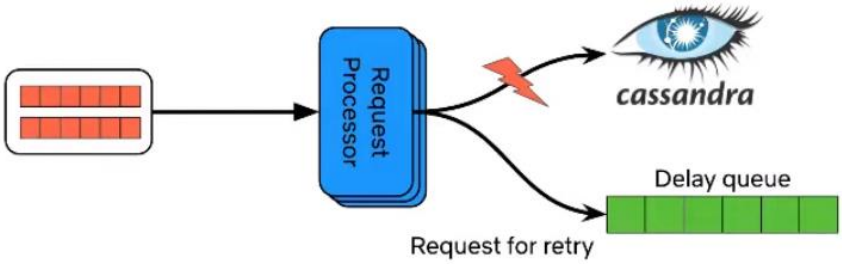
Microservice

Challenges in Async processing

Intermittent processing failures

— Small error rates can be retried asynchronously

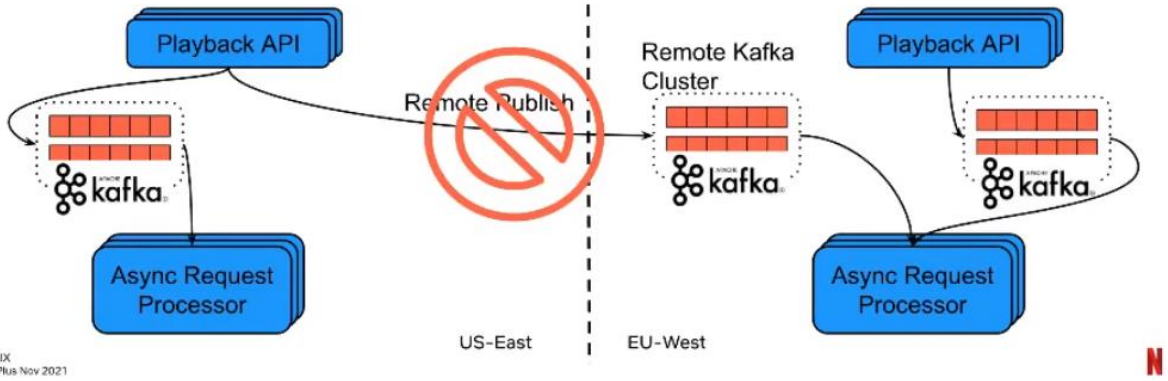
— A “retry” queue



Challenges in Async processing

Cross region routing

— One region for all events of a session
- Region changes during playback

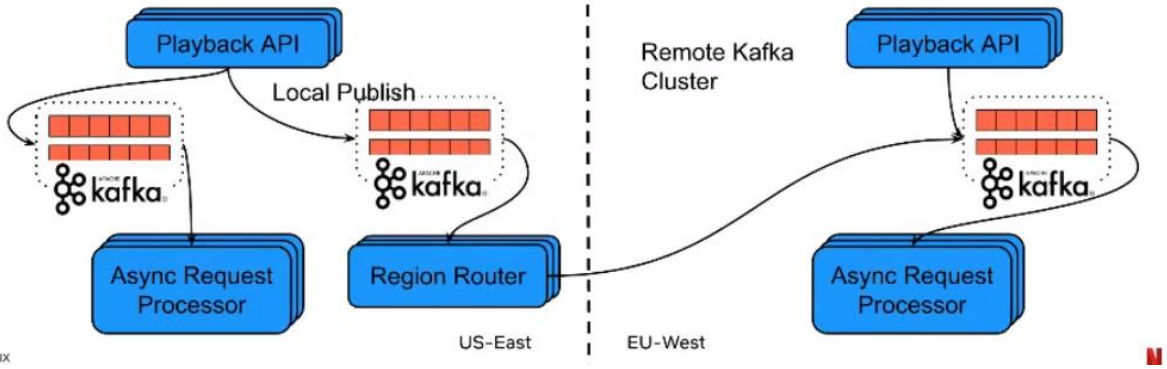


Challenges in Async processing

Cross region routing

— One region for all events of a session
- Region changes during playback

— Local publish with async router

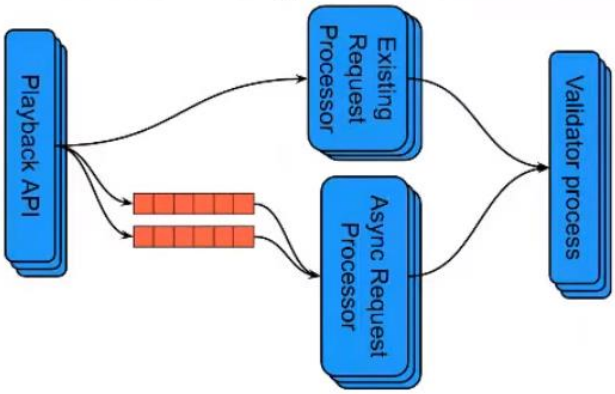


Testing, validation, and rollout

- Shadow testing
- Incremental rollout
- Next steps

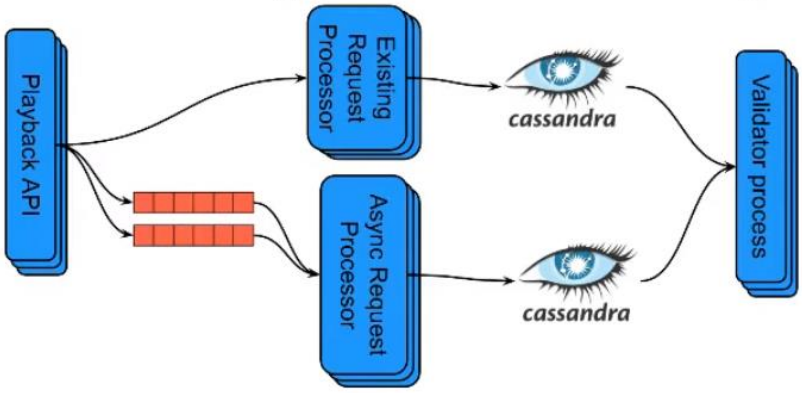
Testing, validation, and rollout

— Shadowing live traffic and storage



Testing, validation, and rollout

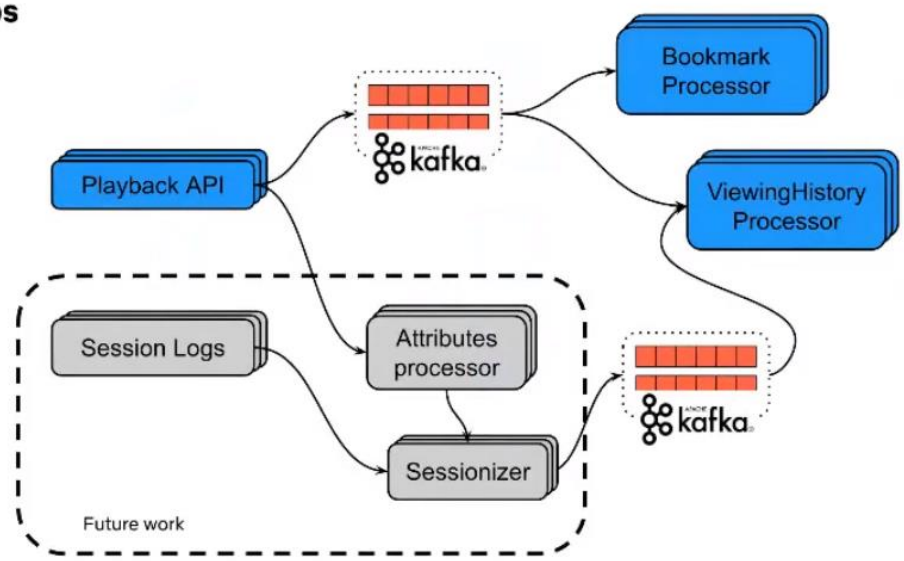
— Shadowing live traffic and storage



— Incremental rollout with “consistent” percentage of traffic

E.g., percentages based on userId

Current rollout and next steps



Conclusion

- Asynchronous processing improved availability and data quality
- Reasoned about design choices and trade offs for async processing
- Shadow testing and incremental rollouts give us a smooth migration