UCI University of California, Irvine

Effect of Caching in a Content-Based Pub/Sub System using Kafka

Distributed Restaurant and Customer Publish/Subscribe System

Mason Nienaber, Radhit Dedania, Tarun Sai Ganesh Nerella

Motivation

- Most distributed service oriented systems use the traditional client-server architecture
  - Every client request handled by remote server, resulting in significant latencies
- Servers typically employ persistent storage techniques for client information in the form of database
  - Database access results in additional significant delay/latency

Objectives

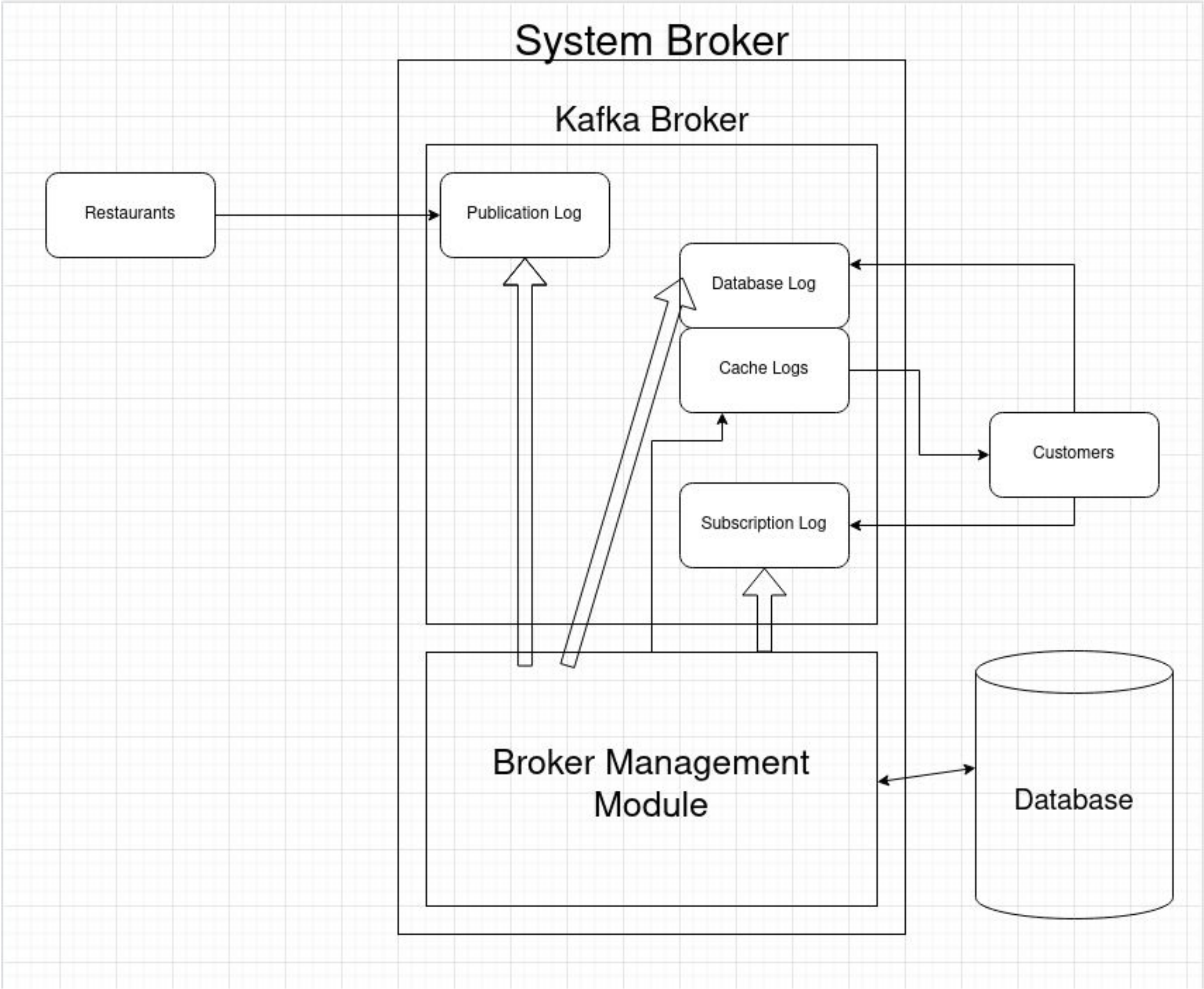
- Implement content-based pub/sub system over Kafka
- Cache messages at broker architecture
  - Due to limited capacity, employ various caching strategies
- Analyze and compare various online cache policies
  - Latency, throughput, hit rate, etc.
- Employ two trivial caching strategies
  - NO caching, unlimited caching
- Employ three novel cache policies
  - Based on number of subscribers to record
  - Threshold, batched average, buffered

Related Works

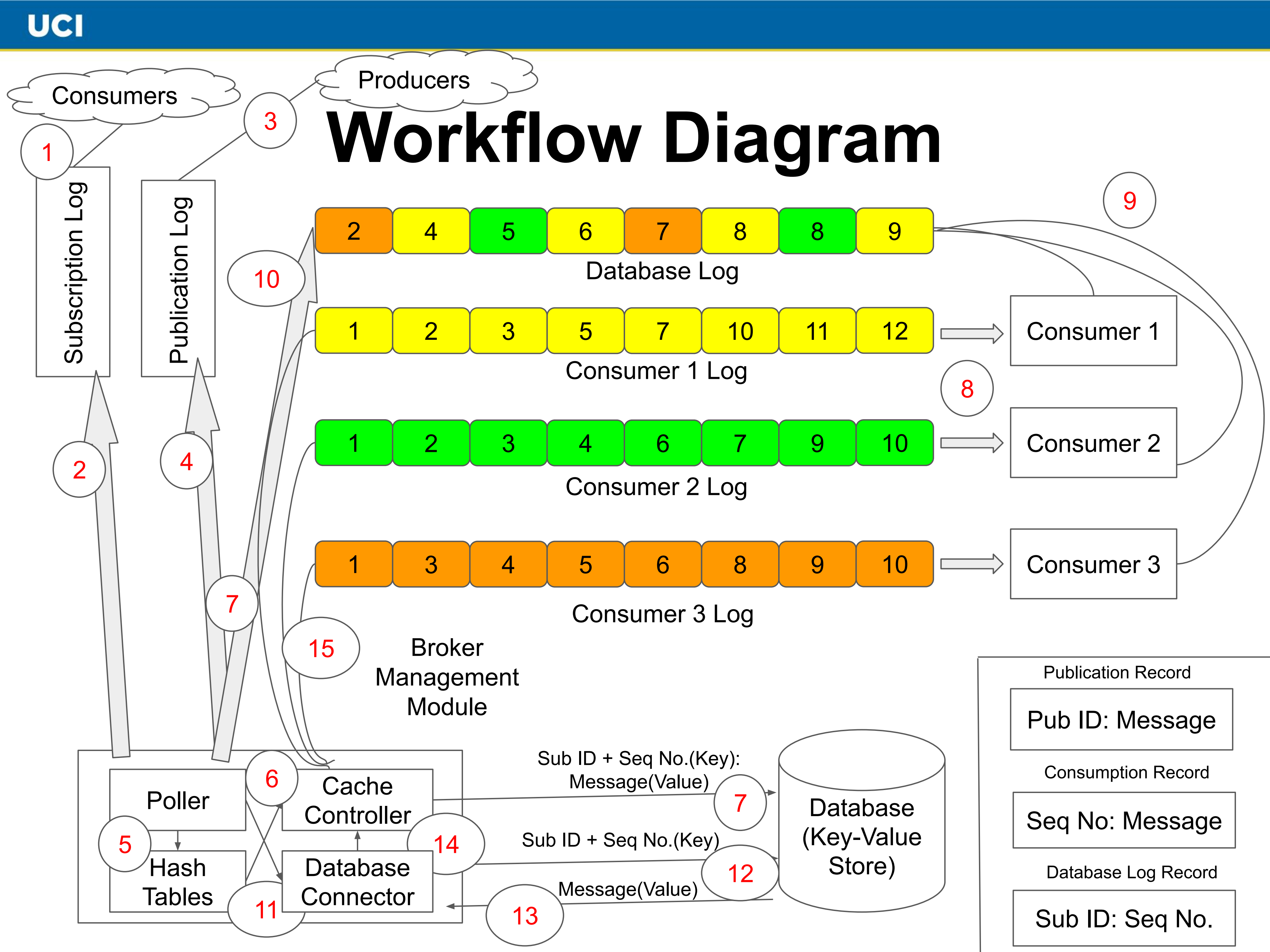
- Notification System
  - Thialfi
  - Siena
- Pub/Sub Systems
  - SpiderCast
  - PolderCast
  - PADRES
  - Cluster-based Pub/Sub
  - Hermes
- Cache Eviction Policy
  - Utility-driven: Maximizing overall sum of utilities(objective function) across subscribers
  - TTL based: sets an expiration time on each object

UCI

System Diagram



Workflow Diagram



Data Structures & Representation

Bitmap(key)	Subscribers(value)
10001	S0,S2
00110	S1

Forward Hash Table

Subscribers(key)	Bitmap(value)
S0	10001
S1	00110
S2	10001

Reverse Hash Table

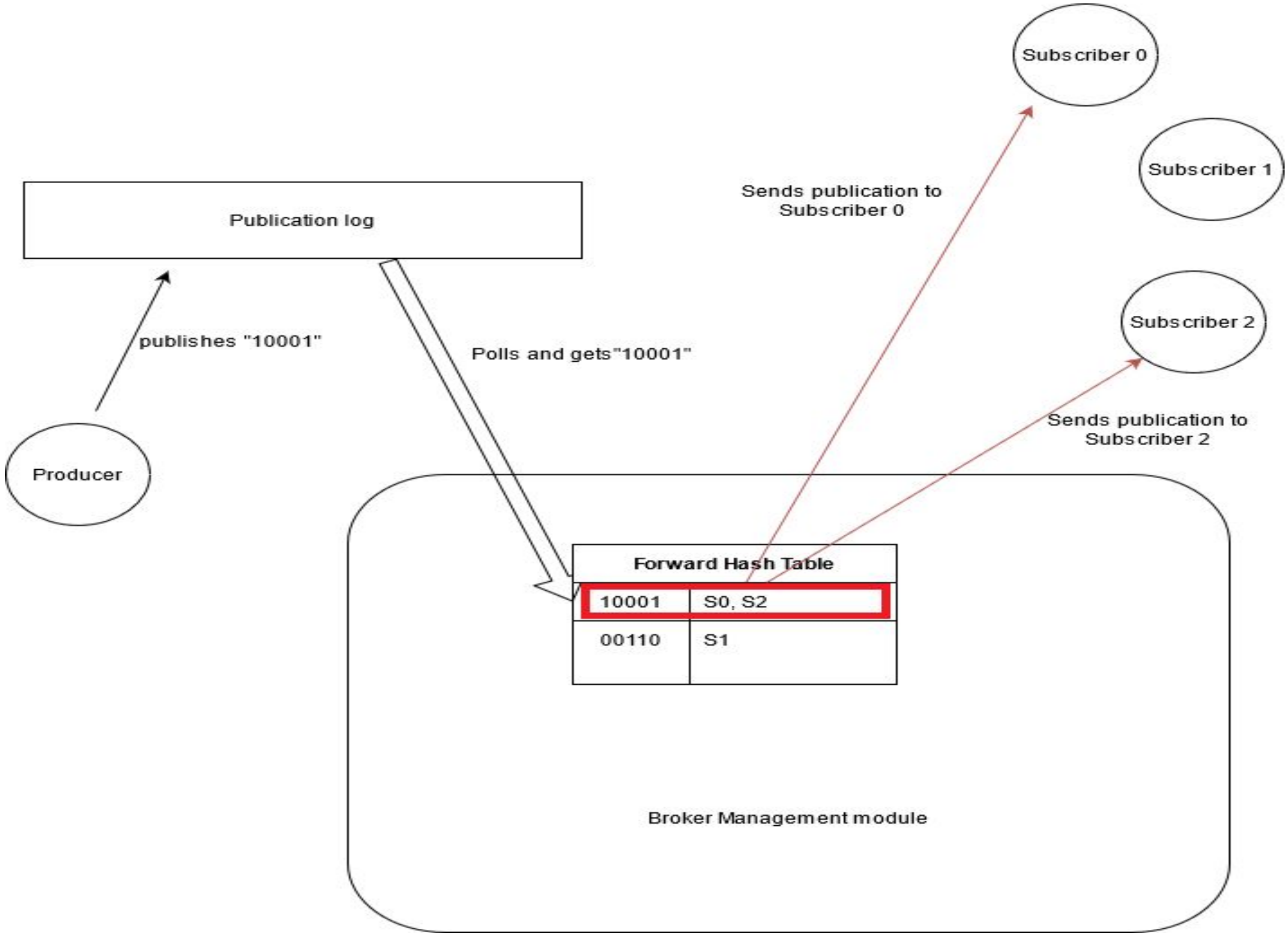
R0 → Restaurant 0 = Dominos
R1 → Restaurant 1 = PizzaHut
R2 → Restaurant 2 = McDonald's
R3 → Restaurant 3 = Taco Bell
F0 → Food item 0 = Burger
F1 → Food item 1 = Pizza
D0 → Discount type 0 = 70% discount
D1 → Discount type 1 = 30% discount
D2 → Discount type 2 = 50% discount

Bitmap	Meaning
10001	R2,F0,D1
00110	R0,F1,D2

BitMap Representation

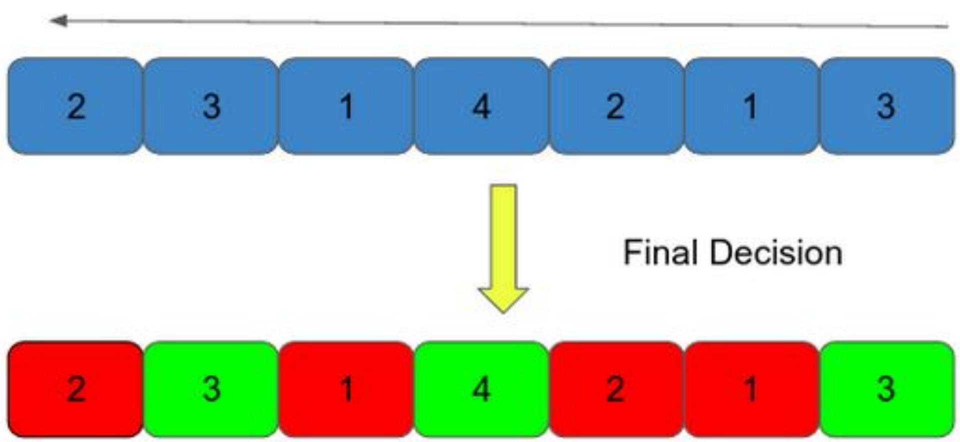
Bitmap 10001 => R2, F0, D1 => McDonald's, Burger, 30% Discount

How Matching occurs



Caching Policies

- *Threshold* Policy



Processing Order

u = utility value of current record = no. of subscribers

λ= 0.5

μ = 0.5

if u > threshold:

send to cache

*Update threshold ← threshold + λ * u*

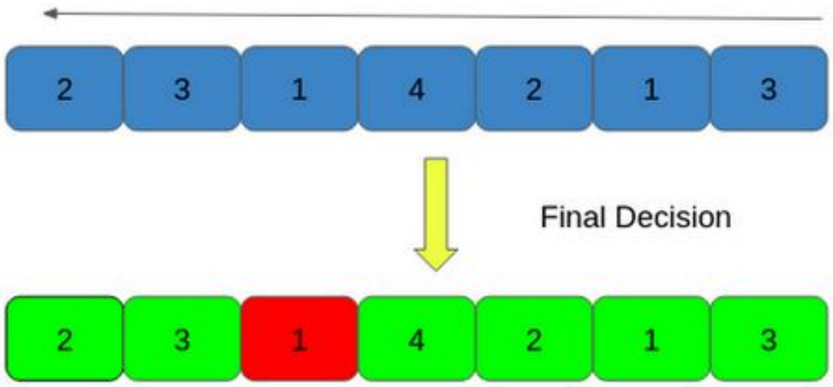
else :

send to database

*Update threshold ← threshold - μ * u*

Caching Policies cont...

- *Mean* Policy



Processing

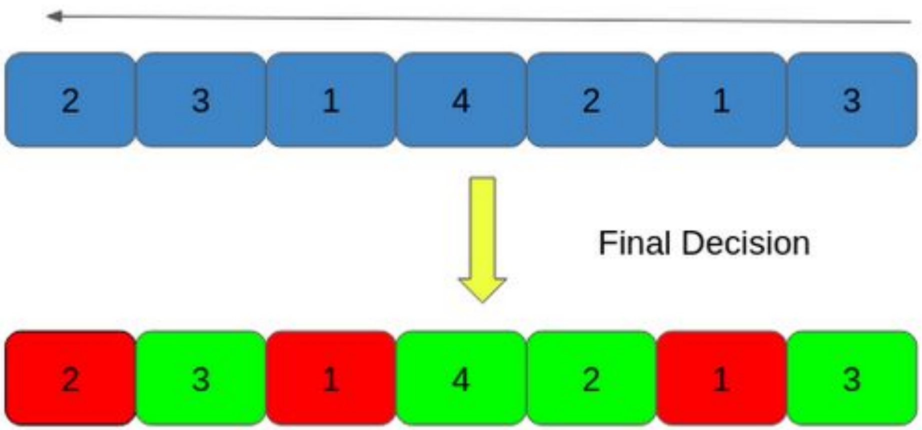
u = current record utility value = no of subscribers
batch_size = 3

batch_sum ← batch_sum + u
*whenever iteration = {batch_size, 2*batch_size, ...}*
update threshold ← (batch_sum/batch_size)
batch_sum ← 0

if (u > threshold)
send to cache
else
send to database

Caching Policies cont...

- *Buffering* Policy



Processing Order

u = current record utility value = no. of subscribers
u_prev = utility value of previous record
$\gamma = 0.7, \alpha = 0.3$

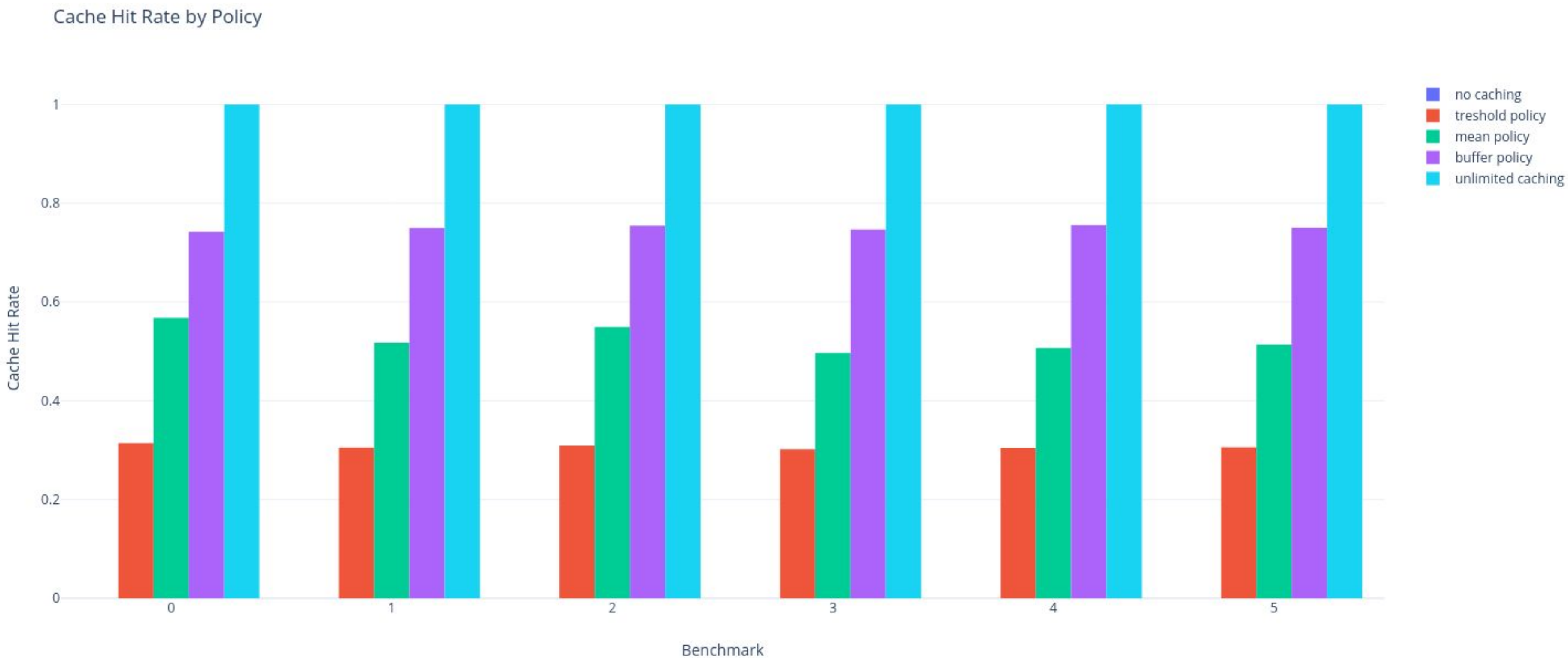
*set threshold ← γ * u + α * u_prev*
if u > threshold
    send to cache
else
    send to database

set u_prev ← u

Testing & Evaluation

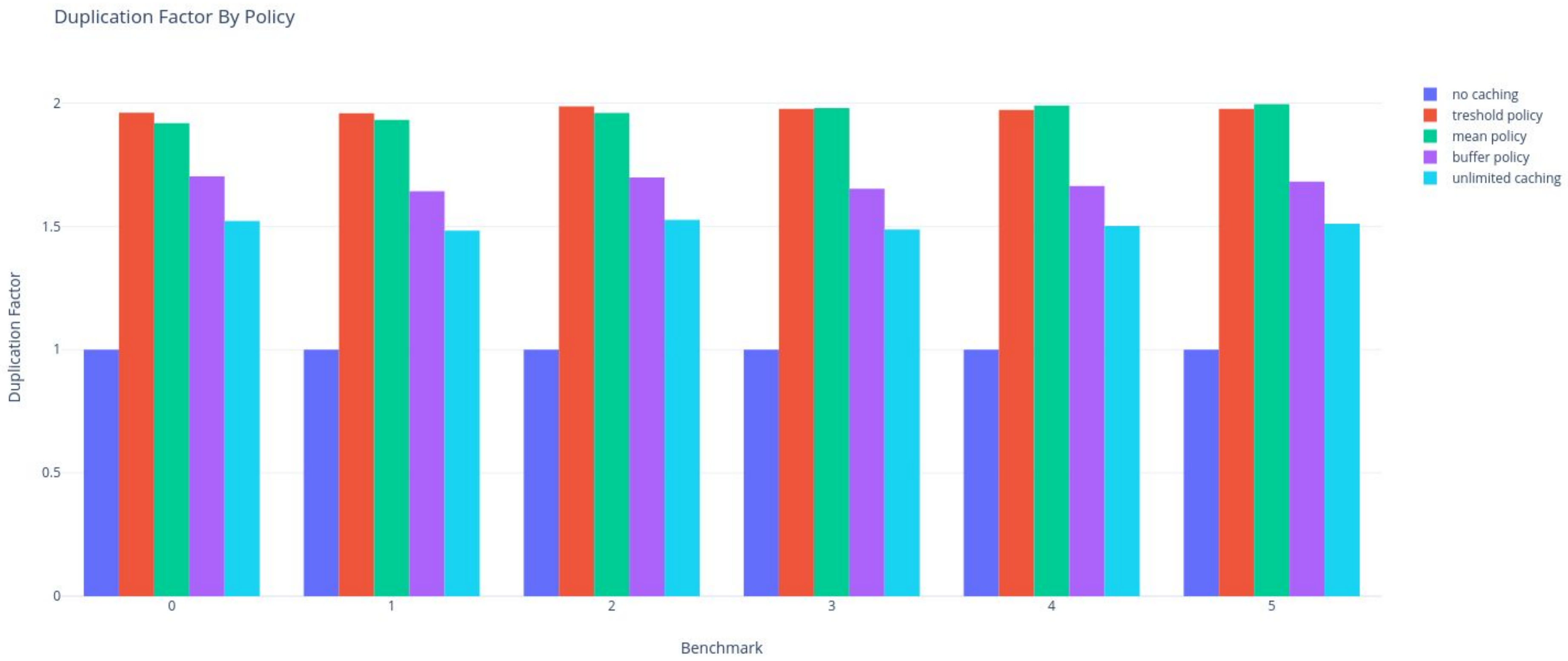
- Seven experiments
  - Four caching metrics
    - Input publication size benchmarks
      - 500,800,1000,2000,5000,10000
    - Cache hit rate, duplication factor, end-to-end latency, throughput
  - Three novel caching policy parameter optimizations
    - Threshold (lambda/mu), Mean (batch size), Buffering (alpha/gamma)

Cache Hit Rate vs. Benchmarks



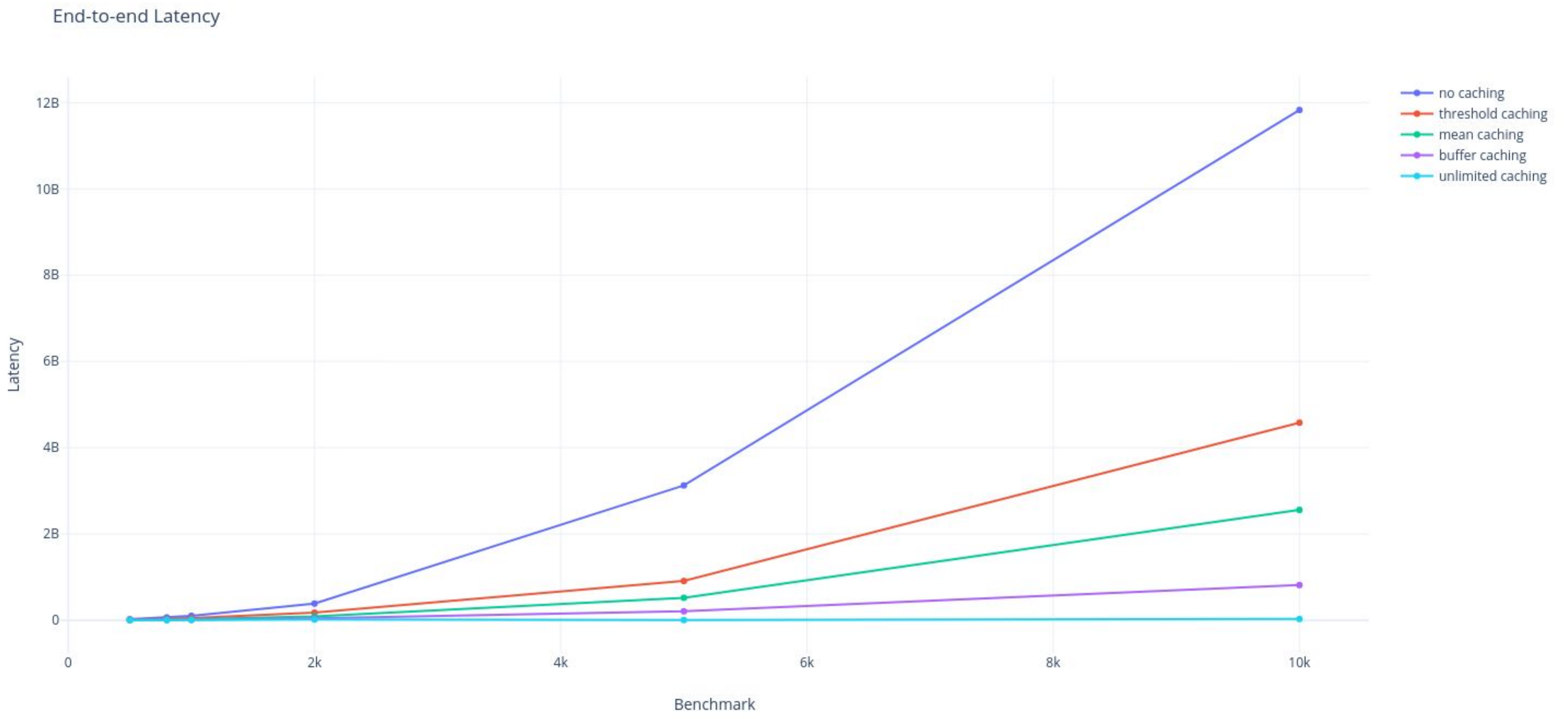
Buffer policy consistently best cache hit rate (excluding unlimited caching) across all policies.

Duplication Factor vs. Benchmarks



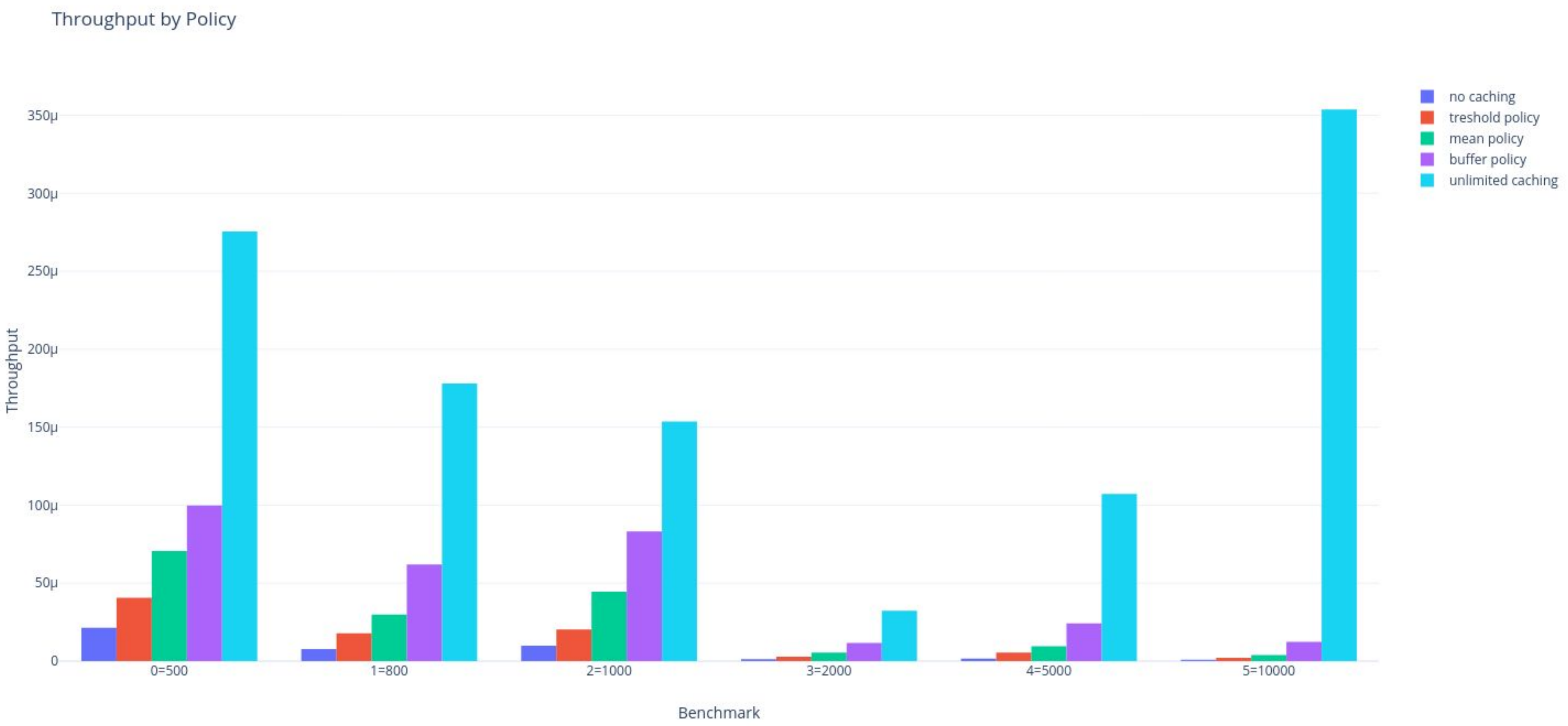
Threshold and Mean policy consistently maintain similar and maximum values across all benchmarks.

Latency vs. Benchmarks



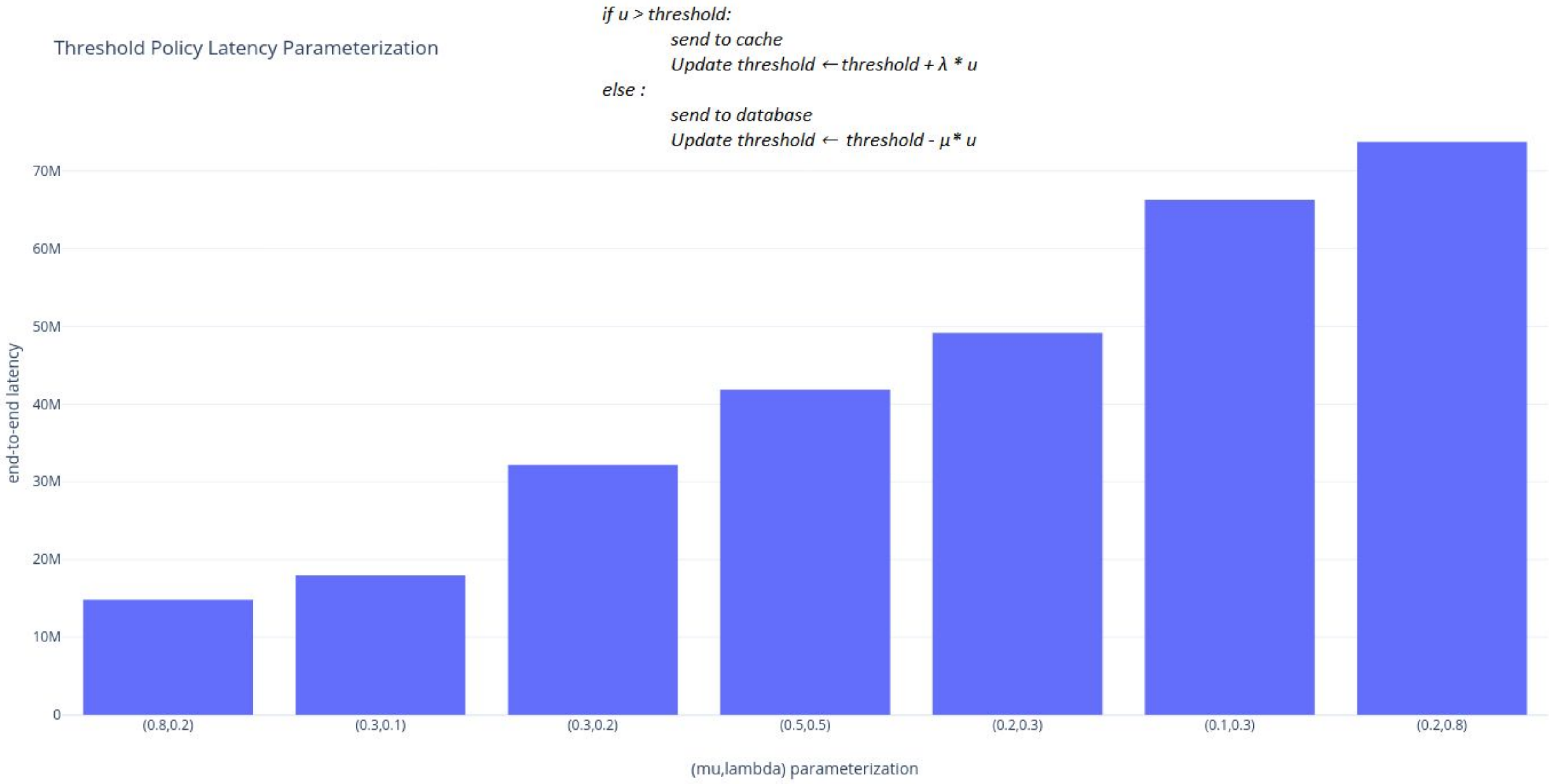
Latency at maximum in no caching, most evident at higher publication size. Negligible latency at low publication size.

Throughput vs. Benchmarks



Throughput at maximum in unlimited caching, most evident at higher publication size.

Threshold Policy Optimization

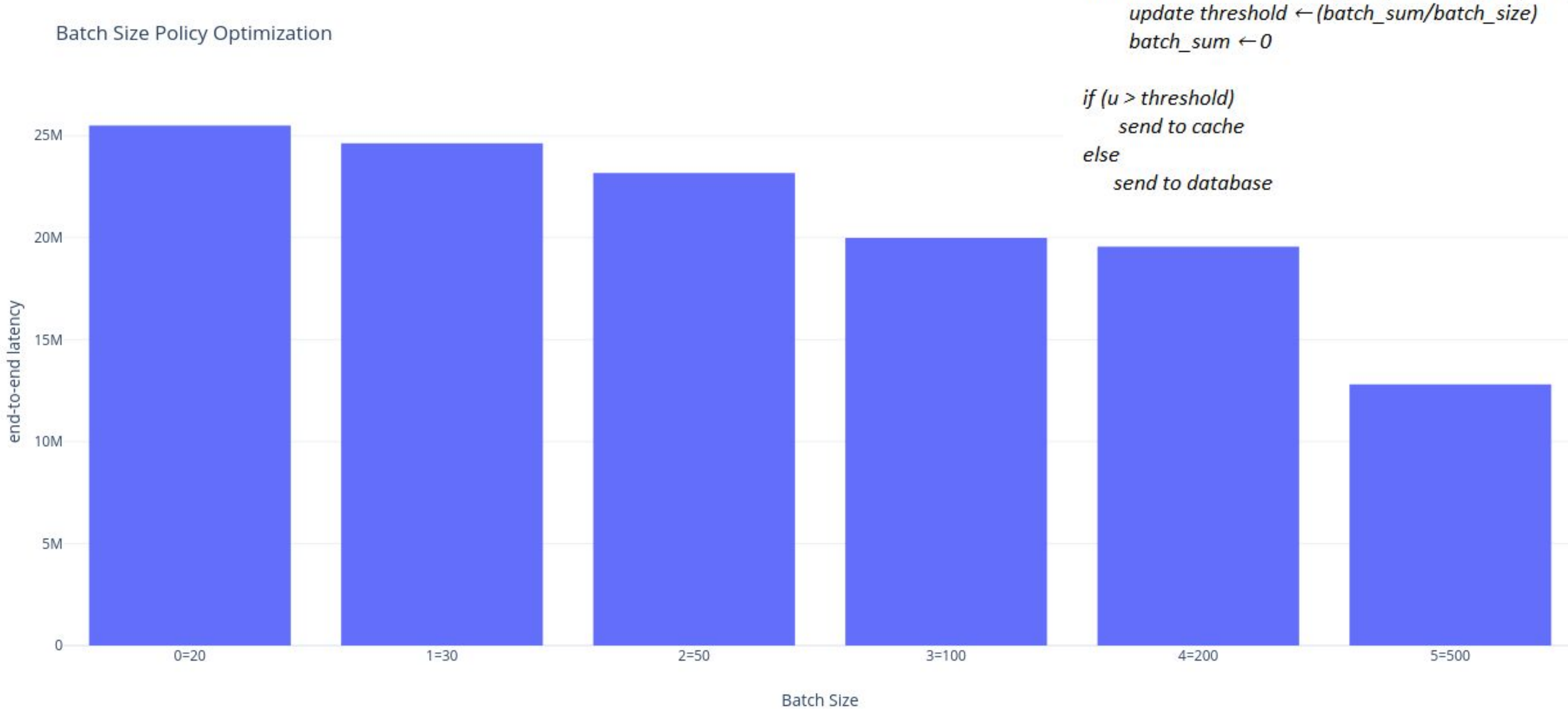


The latency is minimized when mu >> lambda and increases as lambda >> mu.

Mean Policy Optimization

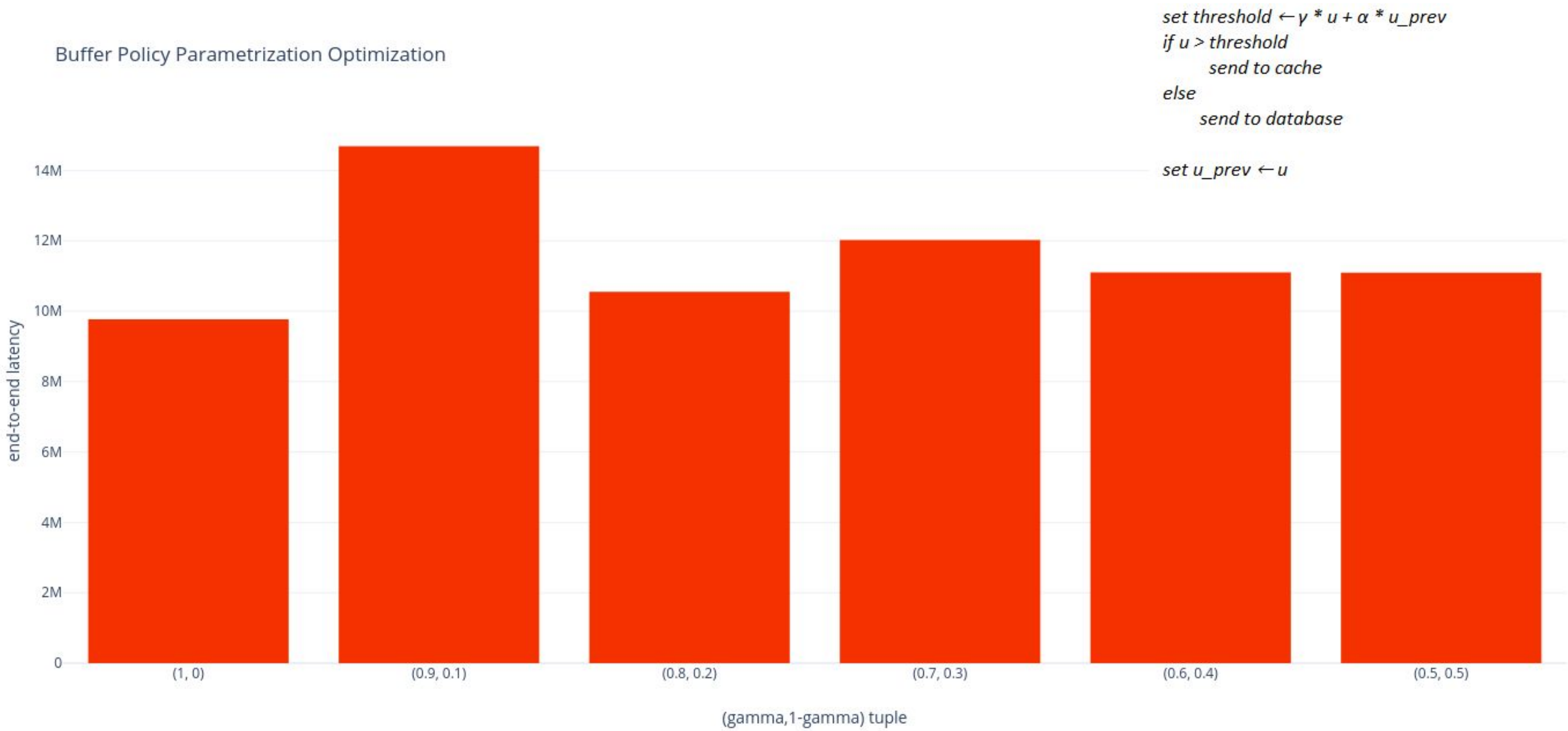
```
batch_sum ← batch_sum + u
whenever iteration = {batch_size, 2*batch_size, .......}
    update threshold ← (batch_sum/batch_size)
    batch_sum ← 0

if (u > threshold)
    send to cache
else
    send to database
```



Latency decreases as batch size increases, implies higher cache hit rate.

Buffering Policy Optimization



Latency is comparably less when threshold update includes a small portion of previous value(subscriber count) which results in variance reduction

Future Work

- Dynamic Addition/Removal of Publishers & Consumers
- Enrichment of Publication Content
- Modification of Subscriptions over time
- Cache Size Aware Caching Policies
- Making Subscriptions Persistent

Conclusion

- Content-Based Pub/Sub over Kafka
- Highlight the benefits of caching at broker for QoS purposes
- Design of three novel caching policies
- Performance comparison across policies
- Scalability with respect to Publishers
- High availability & Fault tolerance (inherently provided by Kafka)

Questions?

Thank you