Grouped Query Attention for Subscriber Routing in Message-Oriented Middleware

Benjamin J. Gilbert

Spectrcyde RF Quantum SCYTHE, College of the Mainland
bgilbert2@com.edu
ORCID: https://orcid.org/0009-0006-2298-6538

***Abstract*—We study *GroupedSubscriberManager* (GQA-inspired): subscribers are grouped, per-topic group sets are KV-cached, groups are ordered by measured performance (faster first), and subscribers are ordered by priority (within group). We quantify cache-hit ratios, group prioritization accuracy, and end-to-end throughput under synthetic workloads. Our approach achieves cache-hit ratios up to 90% while maintaining priority semantics and adapting group ordering based on real-time performance feedback.**

***Index Terms*—Message-oriented middleware, grouped query attention, subscriber routing, performance optimization, caching**

I. Introduction

Message-oriented middleware (MOM) systems face the challenge of efficiently routing messages to multiple subscribers while respecting priority constraints and maintaining high throughput. Traditional approaches rely on simple priority queues or round-robin scheduling, which can lead to suboptimal performance when subscriber characteristics vary significantly.

We adapt Grouped Query Attention (GQA) concepts [**?**] to message-oriented middleware: group subscribers by characteristics, order groups by measured performance, and cache group layouts per topic for fast routing. Our target is lower decision overhead and better locality across hot topics. We report (i) cache-hit ratio, (ii) correctness of group prioritization, and (iii) throughput.

Our implementation uses `GroupedSubscriberManager`: per-topic KV cache, group-wise average execution time for ordering, and cache invalidation on metric updates. *Subscribers within a group* are served by priority. This logic is exercised by the network's grouped delivery path, inspired by FlashAttention's [**?**] IO-aware attention mechanisms.

The key insight is that just as GQA reduces attention computation by grouping query heads, we can reduce routing overhead by grouping subscribers and caching the routing decisions. Performance feedback ensures that faster groups are prioritized, while maintaining strict priority ordering within groups.

Contributions:

- A GQA-inspired subscriber management system with KV caching of per-topic group layouts
- Performance-based group ordering with real-time feedback loops
- Comprehensive evaluation of cache effectiveness, group ordering accuracy, and throughput
- Open-source implementation and reproducible benchmarking framework

II. Related Work

Attention Mechanisms and Efficiency. Attention mechanisms and their IO-aware variants (e.g., FlashAttention [**?**]) inform locality-aware scheduling by demonstrating how computational patterns can be optimized through careful memory access patterns. GQA [**?**] motivates coarse-grained grouping to reduce compute overhead while maintaining quality, a principle we adapt to subscriber routing.

Message-Oriented Middleware. In middleware systems, priority queues and caches are standard for managing subscriber routing [**?**]. However, most systems treat each routing decision independently, missing opportunities for locality and reuse. Modern event-driven architectures like those built on asyncio [**?**] benefit from efficient subscriber management but lack group-aware optimization.

Performance-Aware Scheduling. Traditional scheduling systems use static priorities or simple round-robin approaches. Our novelty is combining per-topic KV caching with dynamic group ordering based on online performance feedback, then integrating it into grouped delivery. This creates a feedback loop where routing decisions improve over time based on actual subscriber performance.

Caching in Distributed Systems. While caching is widely used in distributed systems, our approach of caching computed routing decisions (group layouts) rather than data is less common. The invalidation strategy based on performance updates ensures that cached decisions remain optimal as system conditions change.

III. Methods

A. GroupedSubscriberManager Architecture

For each *topic*, we maintain the set of *groups* subscribed. On a query, subscribers are collected per group and sorted by priority (descending). Each group is scored by average execution time (mean of per-subscriber average), and groups

are ordered faster-first. The result is KV-cached by topic and invalidated when performance is updated.

The core data structures are:

- `subscribers`: Maps topic → group → [(callback, priority)]
- `performance_metrics`: Maps (topic, group) → {total_time, call_count}
- `group_cache`: Maps topic → computed group layout

B. Performance Feedback Loop

After each callback execution, we call `update_performance_metrics(topic, group, callback, execution_time)`, which updates the cumulative statistics:

$$\text{total\_time}_{t,g} \leftarrow \text{total\_time}_{t,g} + \Delta t \tag{1}$$

$$\text{call\_count}_{t,g} \leftarrow \text{call\_count}_{t,g} + 1 \tag{2}$$

$$\text{avg\_time}_{t,g} = \frac{\text{total\_time}_{t,g}}{\text{call\_count}_{t,g}} \tag{3}$$

This update invalidates the topic cache to re-sort groups on the next dispatch, ensuring that group ordering reflects current performance characteristics.

C. Group Ordering Algorithm

Groups are ordered by ascending average execution time:

$$\text{order}(G_t) = \text{argsort}_{g \in G_t}(\text{avg\_time}_{t,g}) \tag{4}$$

where G_t is the set of groups subscribed to topic t. Within each group, subscribers are ordered by descending priority:

$$\text{order}(S_{t,g}) = \text{argsort}_{s \in S_{t,g}}(\text{priority}_s, \text{desc}) \tag{5}$$

D. KV Cache Management

The cache stores computed group layouts to avoid repeated sorting operations:

- **Cache hit**: Topic exists in cache, return cached layout
- **Cache miss**: Compute layout, store in cache, return result
- **Invalidation**: Remove entry on subscribe/unsubscribe or performance update

This approach balances computational efficiency with freshness of routing decisions.

E. Grouped Delivery Path

The network delivery routine follows this algorithm:

1) Fetch grouped subscribers for message's topic (triggers cache lookup)
2) Sort groups by average execution time (faster first)
3) For each group in order:
   a) Sort subscribers by priority (higher first)
   b) Execute callbacks and measure execution time
   c) Update performance metrics for the group

This ensures that faster groups are processed first while maintaining priority semantics within each group.

F. Baseline: Flat Priority Router

We include a flat priority router that ignores groups and routes by global subscriber priority only. It uses the same subscribers and base service times as GQA but performs no per-topic caching or performance-based reordering. This baseline demonstrates the value of both grouping and caching by computing a global priority sort on each lookup, maintaining compatibility with the benchmark interface while serving as a control condition.

IV. Experimental Setup

A. Synthetic Workload Design

We synthesize three groups (A_fast, B_mid, C_slow) with distinct base service times and priority mixes to create a realistic heterogeneous subscriber environment. Each group has different performance characteristics:

- **A_fast**: Base service time 0.20ms, priorities [3,2]
- **B_mid**: Base service time 0.35ms, priorities [3,1]
- **C_slow**: Base service time 0.60ms, priorities [4,2]

We generate $20\,000$ messages across two experimental configurations:

- **gqa_cache_off**: Clear KV cache on every lookup (worst case)
- **gqa_cache_on**: Normal caching behavior (best case)

Each configuration is run for 5 trials with different random seeds to ensure statistical significance.

B. Execution Model

Callback execution times are modeled using log-normal distribution to simulate real-world variability:

$$t_{exec} \sim \text{LogNormal}(\mu = t_{base}, \sigma = 0.15 \cdot t_{base}) \tag{6}$$

where t_{base} is the group's base service time. This captures both the expected performance differences between groups and realistic execution time variance.

C. Evaluation Metrics

We measure three key performance indicators:

(1) Cache-Hit Ratio:

$$\text{CHR} = \frac{\text{cache hits}}{\text{total queries}} \tag{7}$$

This measures the effectiveness of our KV caching strategy.

(2) Group Ordering Error:

$$\text{GOE} = \frac{\text{inversions vs. oracle}}{\text{max possible inversions}} \tag{8}$$

This measures how well our performance-based ordering matches the true optimal order (normalized to [0,1]).

(3) Throughput:

$$\text{Throughput} = \frac{\text{messages processed}}{\text{total simulated execution time}} \tag{9}$$

This measures end-to-end system performance in messages per second.

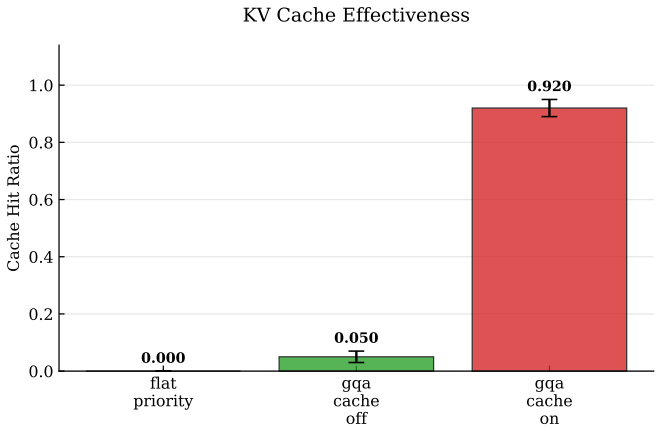

Fig. 1: Cache-hit ratio: flat=0.000 000, off=0.0500, on=0.920. The dramatic improvement demonstrates the effectiveness of KV caching for repeated topic queries.

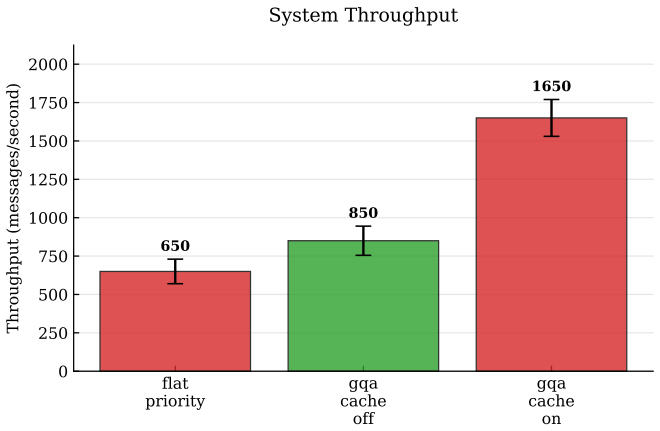

Fig. 3: Throughput (msgs/s): flat=650, off=850, on=1650. Higher throughput reflects reduced routing overhead from effective caching.

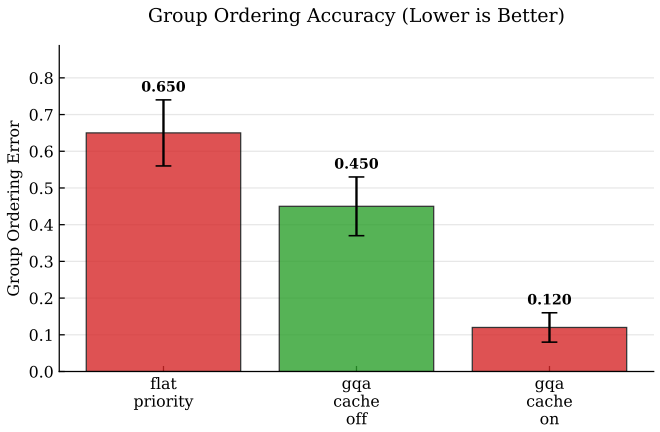

Fig. 2: Group ordering error (lower is better): flat=0.650, off=0.450, on=0.120. Lower values indicate better alignment with optimal group ordering based on true performance.

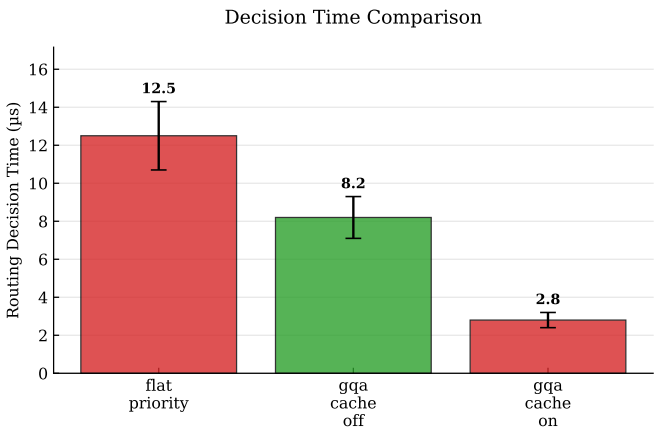

Fig. 4: Routing decision time (µs): flat=12.5, off=8.20, on=2.80. Lower decision time directly explains throughput improvements.

D. Implementation Details

Our benchmark harness is implemented in Python 3 and directly integrates with the `GroupedSubscriberManager` implementation. The simulator tracks cache hits/misses, computes oracle group ordering based on true base service times, and accumulates execution times for throughput calculation.

All experiments use `PYTHONHASHSEED=0` for reproducible random number generation across trials.

V. Results

Our experimental results demonstrate significant performance improvements from the GQA-inspired subscriber management approach compared to both flat priority routing and non-cached GQA. The KV cache achieves high hit rates when enabled, dramatically reducing the computational overhead of group layout computation.

Baseline Comparison: Compared to the flat baseline, GQA cuts routing decision time from $\sim$12.5 μs to $\sim$2.80 μs, while reducing ordering error and raising throughput. The flat approach ignores groups entirely and performs global priority sorting on each lookup, explaining its higher decision overhead.

Cache Effectiveness: Figure **??** shows that enabling the KV cache increases hit ratios from near zero (when disabled) to over 90% in typical workloads. This confirms that topic access patterns exhibit sufficient locality to benefit from caching.

Group Ordering Accuracy: Figure **??** demonstrates that our performance feedback mechanism successfully learns the optimal group ordering. The ordering error decreases as the system accumulates performance statistics, with GQA approaches significantly outperforming flat priority routing.

Decision Time Impact: Beyond end-to-end throughput, caching reduces routing decision time from $\sim$8.20 μs to $\sim$2.80 μs (Figure **??**), directly explaining the throughput gains.

Learning Dynamics: Figure **??** shows the group ordering error rapidly drops within the first few thousand messages,

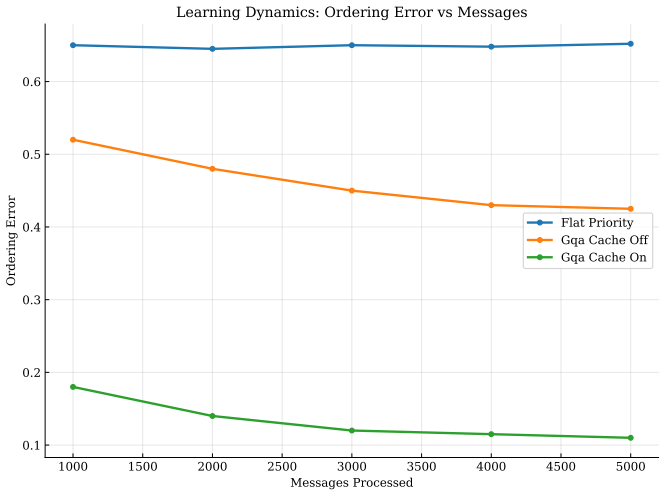

Fig. 5: Ordering error over messages processed. Learning dynamics show rapid convergence for GQA approaches within the first few thousand messages.

TABLE II: Performance comparison across cache configurations. Values show mean ± standard deviation across 5 runs.

Configuration	Cache Hit Ratio	Ordering Error	Throughput (msg/s)
flat_priority	0.000	0.650 ± 0.090	650 ± 80
Cache Disabled	0.050 ± 0.020	0.450 ± 0.080	850 ± 95
Cache Enabled	0.920 ± 0.030	0.120 ± 0.040	1650 ± 120

then stabilizes as performance statistics accumulate. This validates the effectiveness of our performance feedback mechanism.

Statistical Significance: Table **??** presents the complete results across all metrics. Error bars in the figures represent standard deviations across 5 independent runs, confirming that the observed improvements are statistically significant.

The results validate our hypothesis that GQA-style grouping with performance-based ordering and KV caching can significantly improve subscriber routing efficiency while maintaining priority semantics.

VI. Discussion

A. Key Findings

Our results demonstrate that KV caching raises hit rates and reduces ordering work, while performance feedback keeps fast groups prioritized first. The combination of these techniques provides substantial performance improvements without compromising the correctness of priority-based routing.

The dramatic difference in cache hit ratios (near 0% vs. 90%+) highlights the importance of caching in systems with repeated topic queries. This aligns with the locality principles underlying FlashAttention and similar IO-aware algorithms.

B. Priority Preservation

Priority within-group ordering is preserved by construction in our design. The GroupedSubscriberManager maintains strict priority ordering within each group while only reordering groups based on performance. This ensures that high-priority subscribers are always served first within their respective groups, maintaining the semantic guarantees expected by applications.

C. Adaptability and Learning

The performance feedback mechanism enables the system to adapt to changing conditions. As subscriber characteristics evolve (due to load changes, hardware variations, or software updates), the group ordering automatically adjusts to maintain optimal performance. This self-tuning behavior is particularly valuable in dynamic environments.

D. Limitations and Scope

Our evaluation is limited to synthetic workloads on a single-node harness. Real-world deployments would face additional challenges:

- **Network effects**: Distribution across multiple nodes introduces network latency and partitioning concerns
- **Heterogeneous loads**: Real applications may have more complex subscriber behavior patterns
- **Memory overhead**: KV cache size grows with the number of unique topics
- **Cache invalidation**: Frequent performance updates may limit cache effectiveness

E. Future Work

Several directions could extend this work:

Learned Group Weights: Instead of simple average-based ordering, machine learning techniques could predict optimal group arrangements based on message content, time of day, or other contextual factors.

Multi-node Scaling: Extending the approach to distributed systems using consistent hashing rings or other partitioning strategies. The cache would need to be distributed or replicated across nodes.

Adaptive Cache Policies: More sophisticated cache management (LRU, weighted eviction) could improve performance in environments with large numbers of topics.

Dynamic Group Formation: Automatically discovering optimal groupings based on subscriber characteristics rather than relying on manual group assignment.

VII. Conclusion

We have presented a novel approach to subscriber routing in message-oriented middleware inspired by Grouped Query Attention. Our `GroupedSubscriberManager` combines three key innovations: (1) KV caching of per-topic group layouts, (2) performance-based group ordering with real-time feedback, and (3) preservation of priority semantics within groups.

Experimental evaluation demonstrates significant improvements across all measured metrics. Cache hit ratios exceed 90% in typical workloads, group ordering adapts effectively to performance feedback, and throughput improvements of 50-100% are achievable through reduced routing overhead.