Rule Engine Optimization: Caching Initiative

Objective

To improve the performance and scalability of the Core and Common Rule Engine (CCRE) by reducing latency and resource consumption through intelligent caching of:

- Rule definitions
- GraphQL query responses

What We Did

- Implemented in-memory caching for:
  - Parsed Rule Definitions to avoid repeated parsing and validation.
  - GraphQL Query Results for frequently accessed user/segment data.
- Applied cache eviction strategies to maintain memory footprint.
- Ensured thread-safe, concurrent access with no locking overhead.

Key Results

Metric	Before	After	Improveme
P95 Latency	~120 ms	~60 ms	50% reduct
Throughput	~7,000 req/sec	15,000 req/sec	2.1x increa
Memory Footprint	Spiked under load	Stable	No memory
GC Pressure	Moderate-High	Low	Reduced
CPU Utilization	High	Balanced	Improved

Testing Highlights

- Stress-tested in production-like environments simulating regional traffic.
- Sustained 15K rule evaluations per second per region with no degradation.
- Verified zero memory leaks through long-duration runs and heap analysis.
- Compatible with dynamic rule reloading and hot deployments.

Strategic Value

- Improves personalization SLAs across all channels.
- Reduces compute costs by lowering CPU/memory usage.
- Enables horizontal scaling with confidence.

Rule Engine Optimization: Caching Initiative

- Improves developer velocity by simplifying debug/test cycles.
- Strengthens system resilience under load spikes.

Next Steps

- Expand caching to include intermediate evaluation context.
- Introduce distributed cache for multi-region support (e.g., Redis, Hazelcast).
- Build observability dashboards to track rule evaluation metrics per customer/rule.