**Table of Contents:**

1. Why We Need Caching for Gen AI Models
2. Average Caching Hit Rate for Daily Used Gen AI Models
3. Cost Comparison for Caching and Non-Caching Use
4. Use Cases and Their Cache Hit Rate
5. How Technically Caching Works
6. Caching Using Aurora and pgvectors

**1. Why We Need Caching for Gen AI Models**

**Performance Enhancement**

* Reduced Latency: Caching stores frequently accessed data closer to the application, reducing response time.
* Quick Retrieval of Inference Results: Commonly requested prompts or queries are served faster.

**Cost Efficiency**

* Lowering Compute Costs: Reduces the number of times the model is invoked, saving on compute costs.
* Efficient Resource Utilization: Ensures resources are available for more complex and unique queries.

**Scalability**

* Handling High Traffic: Enables the system to handle more requests efficiently during peak times.
* Load Distribution: Reduces the load on AI models and backend infrastructure.

**Improved User Experience**

* Consistency and Reliability: Users receive faster and more consistent responses.
* High Availability: Cached data can be served even during backend service downtimes.

**Enabling Advanced Features**

* Personalization: Quick retrieval of user preferences and previous interactions.
* Interactive Applications: Maintains context and state for interactive AI applications.

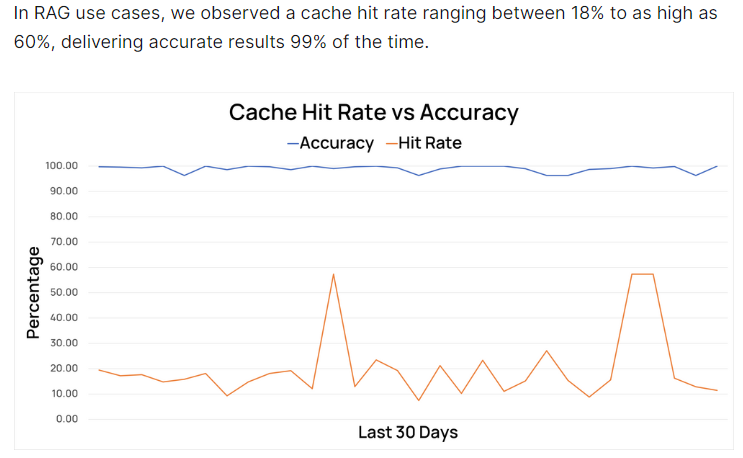
**2. Average Caching Hit Rate for Daily Used Gen AI Models (e.g., ChatGPT-4, Gemini)**

**Influencing Factors**

* Nature of Queries: Higher hit rates for repetitive queries; lower for unique queries.
* Cache Size and Policy: Larger cache size and effective eviction policies improve hit rates.
* User Behavior: Consistent user behavior leads to higher hit rates.
* Time-to-Live (TTL): Longer TTL increases hit rates.

**Typical Hit Rates**

* **General Average: 60-90%**.



Example:

**3. Cost Comparison for Caching and Non-Caching Use**

**Without Caching**

* Request Volume: 10,000 requests per day.
* Cost per Inference: $0.10.
* Total Cost: 10,000 x $0.10 = $1,000 per day.

**With Caching**

* Cache Hit Rate: 70%.
* ElastiCache Cost: $0.03 per hour per node.
* Requests Served from Cache: 7,000 (70%).
* Requests Not Served from Cache: 3,000 (30%).
* Cost for Non-Cached Requests: 3,000 x $0.10 = $300.
* ElastiCache Cost per Day: $0.03 x 24 = $0.72.
* Total Cost with Cache: $300 + $0.72 = $300.72 per day.

**Cost Savings**

* Daily Savings: $1,000 - $300.72 = $699.28.
* Monthly Savings: $699.28 x 30 = $20,978.40.
* Annual Savings: $699.28 x 365 = $255,235.20.

**4. Use Cases and Their Cache Hit Rate**

**High Hit Rate Use Cases**

* Customer Support Chatbots: 80-90%.
* FAQ Systems: 85-90%.

**Moderate Hit Rate Use Cases**

* Content Generation Platforms: 60-75%.
* Interactive Storytelling Applications: 50-70%.

**Lower Hit Rate Use Cases**

* Creative Writing Prompts: 50-60%.
* Exploratory Data Analysis Tools: 55-70%.

**5. How Technically Caching Works**

**In-memory caching** stores data in the memory (RAM) of a server or cluster, allowing for extremely fast data retrieval.

**Persistent caching** stores data on disk, allowing it to survive restarts, power failures, and other interruptions.

**Cache Storage**

* In-Memory Caching: Using services like Amazon ElastiCache (Redis, Memcached).
* Persistent Caching: Using databases like Amazon Aurora with pgvector for vector similarity searches.

**Cache Retrieval**

* Lookup Mechanism: Checking the cache for requested data before querying the model.
* Cache Eviction Policies: Strategies like Least Recently Used (LRU) to manage cache storage.

**Cache Updates**

* Time-to-Live (TTL): Defines how long data stays in the cache.
* Invalidation Policies: Determines when to remove or refresh cached data.

**Cost Comparison: In-Memory vs. Persistent Caching on AWS**

|  |  |  |
| --- | --- | --- |
| Feature | In-Memory Caching | Persistent Caching |
| Service | Amazon ElastiCache (Redis or Memcached) | Amazon RDS with Aurora |
| Example Instance Type | cache.t3.medium | t3.small |
| Cost per Hour | $0.031 | $0.050 |
| Cost per Day | $0.031 × 24 = $0.744 | $0.050 × 24 = $1.20 |
| Cost per Month | $0.744 × 30 = $22.32 | $1.20 × 30 = $36 |
| Data Durability | Volatile (data lost on restart) | Persistent (data survives restarts and failures) |
| Latency | Very low (RAM-based) | Slightly higher (disk-based or managed cache) |
| Primary Use Case | Fast, low-latency access to frequently accessed data | Durable storage with good performance |
| Additional Costs | Backup and data transfer | Data transfer |

**6. Caching Using Aurora and pgvectors**

**Amazon Aurora with pgvector**

* Vector Similarity Searches: Storing embeddings and performing efficient retrievals.
* Persistent Caching: Suitable for more complex queries and long-term storage.

**Implementation Steps**

1. Setting Up Aurora: Configure Amazon Aurora for your application.
2. Integrating pgvector: Install and use the pgvector extension for vector-based caching.
3. Storing Embeddings: Store embeddings generated by the AI model in Aurora.
4. Querying Embeddings: Use vector similarity searches to retrieve relevant cached data.

Ref:

https://aws.amazon.com/bedrock/pricing/