ElastiCache For RDS

1 Introduction
- 1 Imagine a ride-sharing app like Uber that connects passengers to nearby drivers.
- 2 For the app to work smoothly
  - 1 It needs to quickly find the nearest driver
  - 2 Update driver statuses in real time
  - 3 Handle thousands of users without delays
- 3 Traditionally, this would rely on a database like RDS (Relational Database Service), but RDS alone struggles to handle such real-time needs efficiently during high traffic
- 4 To solve this, we introduce ElastiCache, a caching layer that uses
  - 1  Redis
  - 2  Memcached
- 5 Let’s understand how ElastiCache improves performance and determine which option, Redis or Memcached, is best suited for Uber’s use case

2 How ElastiCache Enhances Uber’s Performance
- 1 Real-Time Driver Matching
  - 1 RDS Only
    - 1 Driver locations are stored in RDS as latitude and longitude in a table
    - 2 To find the nearest driver, RDS calculates the distance for every driver in the table using complex mathematical formulas
    - 3 The process is slow and resource-intensive, especially with thousands of drivers
    - 4 As traffic increases, RDS struggles to provide real-time results.
  - 2  RDS + Redis
    - 1 Redis uses a geospatial index to store driver locations
    - 2 It quickly finds nearby drivers using optimized queries
    - 3 Redis provides results in milliseconds, ensuring instant responses even during peak traffic
    - 4 Offloads heavy geospatial queries from RDS, reducing its load
  - 3  RDS + Memcached
    - 1 Memcached cannot handle geospatial data
    - 2 It lacks the ability to store and query location-based data efficiently
- 2 Caching Frequent Searches
  - 1 RDS Only
    - 1 Passengers frequently search for drivers in popular areas like downtown
    - 2 RDS repeatedly processes the same queries, increasing database load
    - 3 This slows down performance for other queries, leading to delays
  - 2  RDS + Redis
    - 1 Redis caches the results of frequent searches in memory
    - 2 If the same query is made again, Redis instantly provides the result
    - 3 This reduces the load on RDS and speeds up response times
  - 3  RDS + Memcached
    - 1 Memcached can also cache query results and retrieve them quickly
    - 2 It works well for simple caching but doesn’t support structured or complex data updates like Redis
- 3 Real-Time Notifications
  - 1 RDS Only
    - 1 Notifications for drivers rely on periodic checks or external systems
    - 2 Drivers don’t receive updates instantly, causing delays in ride acceptance
  - 2  RDS + Redis
    - 1 Redis uses Pub/Sub (Publish/Subscribe) to send real-time notifications
    - 2 When a passenger requests a ride, Redis instantly notifies drivers subscribed to the notification channel
    - 3 This ensures real-time communication between passengers and drivers
  - 3  RDS + Memcached
    - 1 Memcached does not support real-time messaging or Pub/Sub
    - 2 Notifications cannot be implemented using Memcached
- 4 Tracking Active Drivers
  - 1 RDS Only
    - 1 Active drivers are stored in a table in RDS
    - 2 Every time a driver logs in or out, the table is updated
    - 3 Frequent updates slow down RDS, especially during high traffic
  - 2  RDS + Redis
    - 1 Redis tracks active drivers in memory using hashes or sets.
    - 2 Driver status updates (e.g., online/offline) are instant
    - 3 Redis reduces database writes and provides real-time tracking
  - 3  RDS + Memcached
    - 1 Memcached can store basic driver statuses but lacks advanced features
    - 2 It cannot handle structured data like Redis, making it less flexible
- 5 Driver Leaderboard
  - 1 RDS Only
    - 1 RDS ranks drivers based on performance metrics like completed rides or ratings
    - 2 Sorting large datasets in real time requires heavy computations
    - 3 Rankings take longer during high traffic, delaying updates
  - 2  RDS + Redis
    - 1 Redis uses sorted sets to maintain a real-time leaderboard
    - 2 Scores (e.g., completed rides) are updated instantly, and rankings are automatically sorted
    - 3 This ensures real-time leaderboards without additional computation
  - 3  RDS + Memcached
    - 1 Memcached doesn’t support sorted sets or ranking features
    - 2 Leaderboards cannot be implemented using Memcached
- 6 Data Persistence
  - 1 RDS Only
    - 1 RDS stores all data permanently on disk, ensuring reliability
    - 2 However, high traffic and constant updates slow down performance
  - 2  RDS + Redis
    - 1 Redis operates in memory but supports optional persistence
    - 2 Critical data like driver statuses or cached search results can be saved to disk
    - 3 This combines the speed of Redis with the reliability of RDS
  - 3  RDS + Memcached
    - 1 Memcached is memory-only and does not support persistence
    - 2 All cached data is lost if the system restarts

3 Conclusion
- 1 RDS Only Great for storing structured data but struggles with real-time performance and scalability during high traffic
- 2  RDS + Redis Redis supports advanced features like geospatial indexing, real-time notifications, and sorted sets for leaderboards, making it perfect for complex, real-time use cases.
- 3  RDS + Memcached
  - 1 Use Memcached when your caching needs are simple, and the data doesn’t require advanced features like
    - 1 Geospatial queries
    - 2 Real-time messaging
    - 3 Persistence
  - 2 It is ideal for applications that require lightweight, high-speed caching without complex data operations.
  - 3 Memcached is also perfect for temporary storage where losing data on restart is acceptable