SYSTEMS DESIGN TASK SOLUTIONING – DSTREET GAMES

By Pooja Bennabhaktula

Rough Picture

Scaling application as it grows popularity is a critical task!
As users grow, we want to reduce the load on databases and use a cache to avoid re computations (as in we don't want to recompute the relative leader board scores for every single request every single time) and reduce DB attacks/blockage when there are multiple requests. So, in order to deliver faster response time to your client you need a cache.

Solutioning

For the given task an ***in-memory database*** provides a better solution than traditional databases that store data on disk or SSDs. Access time is vastly reduced when in memory DB is employed. So, our objective of delivering sorted results in real-time goes something like this.

So, in the context with using Amazon Web Services for our problem, we have two options:

- ***Redis***
- ***DynamoDB+ Local Secondary Index***

DynamoDB is inexpensive compared to Redis, and if considered according to the ***assumption*** "Let us assume that, at any point of time there are 100K pools, each with entries ranging from 1000 to 500,000.", given in the documentation. DynamoDB serves the purpose.

Pros of the Design

- Relatively inexpensive
- Stores information with high performance algorithms, in B trees and hashing.
- Amicable with AWS ecosystem
- Fully Managed by AWS
- DynamoDB offers a free tier which allows up to 40 million database operations a month for free, which means we can check-this out and migrate to Redis if there is any inconsistency!

Cons of the Design

- Can create Hot key problem
- DynamoDB does not support creating large hash-keys
- Poor documentation

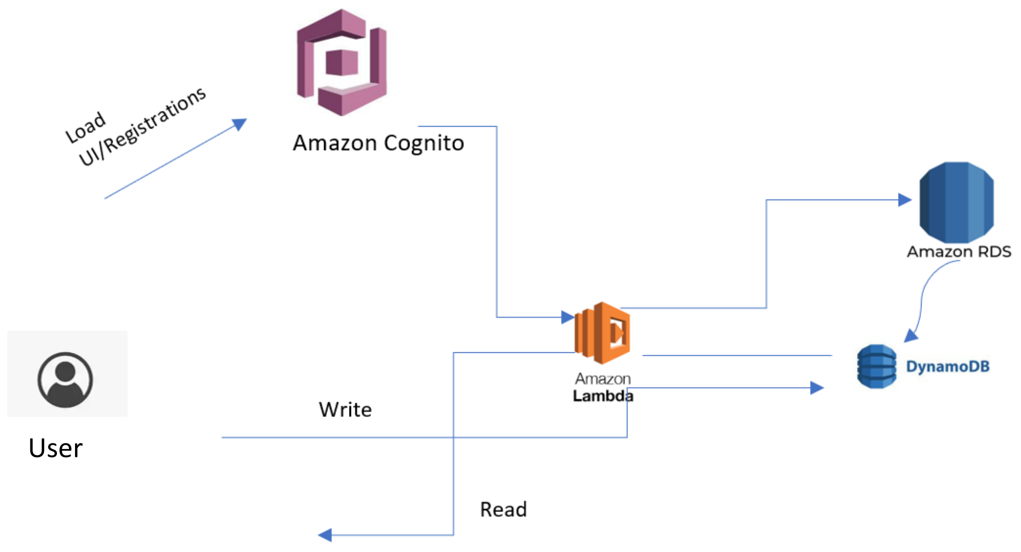

Fig 1.1- Solutioning with DynamoDB

Putting all the pieces together,

- A user will sign up & create a new account (Amazon Cognito)
- After creation of account the second end point is the use of client to receive an ID (Amazon Lambda)
- User will use the add user score end point to introduce score for user (Amazon RDS)
- Ranking is fetched accordingly. (Dynamo DB + DAX)

Under certain circumstances, of fixed small number of users (say 10 pools) this solution is an overkill.

Note:

However, if cost is not an issue and users are growing exponentially, **Redis** is the rescue as it provides blazing hot accuracy and has god-speed potential :D (Cons- Risk of losing data when failure)

References:

https://medium.com/fernando-pereiro/caching-strategy-with-dynamodb-streams-aws-lambda-and-elasticache-2b309333cff9
https://aws.amazon.com/blogs/opensource/aws-service-operator-kubernetes-available/
https://d1.awsstatic.com/whitepapers/aws-scalable-gaming-patterns.pdf
https://aws.amazon.com/blogs/database/how-to-use-dynamodb-global-secondary-indexes-to-improve-query-performance-and-reduce-costs/
https://aws.amazon.com/blogs/apn/how-to-build-a-real-time-gaming-leaderboard-with-amazon-dynamodb-and-rockset/
https://stackoverflow.com/questions/26514730/how-to-model-a-highscore-with-1-million-players-in-dynamodb-and-prevent-a-hot-ke
’https://hackernoon.com/redis-gamification-60e49b5494ae