CLOUD COMPUTING APPLICATIONS

Cloud Caching Strategies

Prof. Reza Farivar

Caching Strategies

- strategies to implement for populating and maintaining your cache depend upon what data you cache and the access patterns to that data
  - Cache Aside (Lazy Loading)
  - Write-Through
  - Adding TTL
- Deploying nodes to multiple Availability Zones (Elasticache supports this) can avoid single point of failure and provides high availability

The Right Caching Strategy

- Cache-Aside (Lazy Loading)
  - Applicating data is written only into the source
  - Only loads data to the cache when it is required on a “read”
    - Typically most data is never requested
  - Suitable for read-heavy Applications
  - Allows stale data
  - In case of cache node failure, just read from source
- Write-Through
  - Applicating data is written into the cache and source at the same time
  - Suitable for write-heavy Applications, where data loss is not acceptable
    - But every write is expensive
  - Cache never gets stale
  - In case of cache node failure, just read from source
- Write-Back (Lazy Writing)
  - Applicating data is written only to the Cache
  - More complex to implement

Cache Sharding

- There is only one machine that contains each piece of data
- Memcached:
  - Consistent Hashing ring algorithm
- Redis:
  - Elasticache for Redis can have up to 250 shards
  - Each shard can consist of a master Redis node, and up to 5 Redis Read replicas
- Sharding is a great technique but has its own problems
  - Resharding data when adding/removing nodes
  - Celebrity problem
  - Join and de-normalization:
    - Not as big a problem in Caches, but can be serious for databases
    - Once data is sharded, it is hard to perform join operations across all shards.

Time to Live (TTL)

- Lazy loading allows for stale data but doesn't fail with empty nodes
- Write-through ensures that data is always fresh, but can fail with empty nodes and can populate the cache with superfluous data
- By adding a time to live (TTL) value to each write, you can have the advantages of each strategy. At the same time, you can and largely avoid cluttering up the cache with extra data.
- *Time to live (TTL)* is an integer value that specifies the number of seconds until the key expires
  - Redis can specify seconds or milliseconds for this value.
  - For Memcached, it is seconds.
- When an application attempts to read an expired key, it is treated as though the key is not found. The database is queried for the key and the cache is updated.
- This approach doesn't guarantee that a value isn't stale. However, it keeps data from getting too stale and requires that values in the cache are occasionally refreshed from the database.