Round-trips back and forth to a database and its underlying storage can add significant delays and are often the top contributor to application latency.

Caching frequently-used data is one of the most important performance optimizations you can make in your applications

querying a database is an expensive operation compared to in-memory.

read-heavy applications.

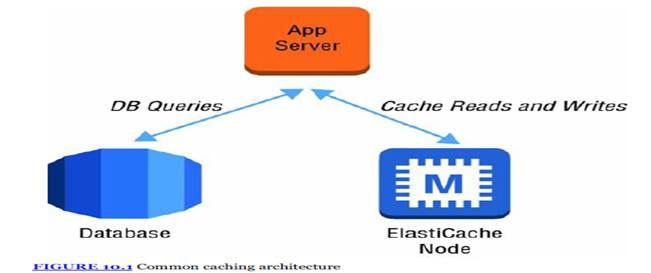
Example – application session state for a large website can be stored in an in-memory caching engine, instead of storing the session data in the database

**Caching Pattern**

The app server checks the cache first to see if it contains the data it needs.

If the data does not exist in the cache node, it will query the database and serialize and write the query results to the cache.

The next user request will then be able to read the data directly from the cache instead of querying the database.



**Data Access Patterns**

Which data should be cached ?

You should evaluate the access pattern of the data before you decide to store it in cache.

Example : List of products in a catalog.. candidate for cache… accessed thousand of times per second..

**Cache Engines**

**Memcached**

provides a very simple interface that allows you to write and read objects into in-memory key/value data stores

You can partition your cluster into shards and support parallelized operations for very high performance throughput.

Memcached deals with objects as blobs that can be retrieved using a unique key.

R**eddis**

Can be used as cache + DB + Message broker

Redis supports a rich set of data types likes strings, lists, and sets + Objects

Redis supports the ability to persist the in-memory data onto disk. This allows you to create snapshots that back up your data and then recover or replicate from the backups.

Support up to five read replicas to offload read requests.

failure of the primary node, a read replica can be promoted and become the new master using Multi-AZ replication groups

advanced features that make it easy to sort and rank data.

**Nodes and Clusters**

Each deployment of Amazon ElastiCache consists of one or more *nodes* in a *cluster*

A single Memcached cluster can contain up to 20 nodes

Redis clusters are always made up of a single node; however, multiple clusters can be grouped into a *Redis replication group*.

Design for Failure

While it is unlikely, you should plan for the potential failure of an individual cache node.

For Memcached clusters, you can decrease the impact of the failure of a cache node by using a larger number of nodes with a smaller capacity, instead of a few large nodes.

**Scaling**

**Horizontal Scaling**

With Memcached, you can partition your data and scale horizontally to 20 nodes or more. With Auto Discovery, your application can discover Memcached nodes that are added or removed from a cluster

Multiple clusters can be grouped into replication group

can have up to five read replicas handling read-only requests.

**Vertical Scaling**

Support limited

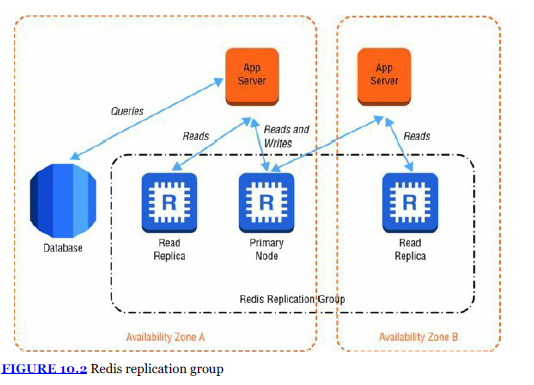
New clusters can be created

                For memcached new cluster is empty

Redis cluster can be initialized from a backup

**Replication and multi AZ**

Replication is asynchronous and there is some delay



**Backup and Recovery**

Redis -- > DB and snapshots

Memchached – not possible

Use a combination of automatic and manual snapshots to meet your recovery objectives for your Redis cluster.

Memcached is purely in-memory and does not have native backup capabilities.

**When to use Memcache v/s Redis**

Use Memcached when you need a simple, in-memory object store that can be easily partitioned and scaled horizontally.

Use Redis when you need to back up and restore your data, need many clones or read replicas,

You are working on a mobile gaming application and are building the leaderboard feature to track the top scores across millions of users. Which AWS services are best suited for this use case?

Answer - Amazon ElastiCache using Redis

A. Configure AutoDiscovery on the client side

1 - ABC

2 BC

3 C

4 A

5 B

6 BC

7 BCD

8 A

9 B

10 AB

Thanks & Regards,

Shantaram Vernekar