# System Design: The Distributed Cache

Learn the basics of a distributed cache.

We'll cover the following



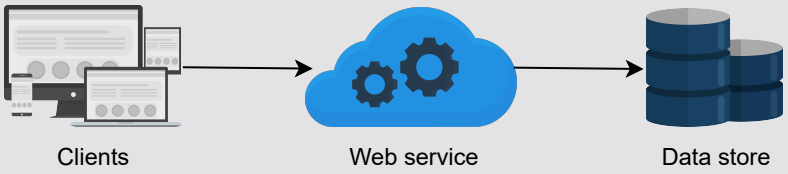
## Problem statement

A typical system consists of the following components:

- It has a client that requests the service.
- It has one or more service hosts that entertain client requests.
- It has a database used by the service for data storage.

Under normal circumstances, this abstraction performs fine. However, as the number of users increases, the database queries also increase. And as a result, the service providers are overburdened, resulting in slow performance.

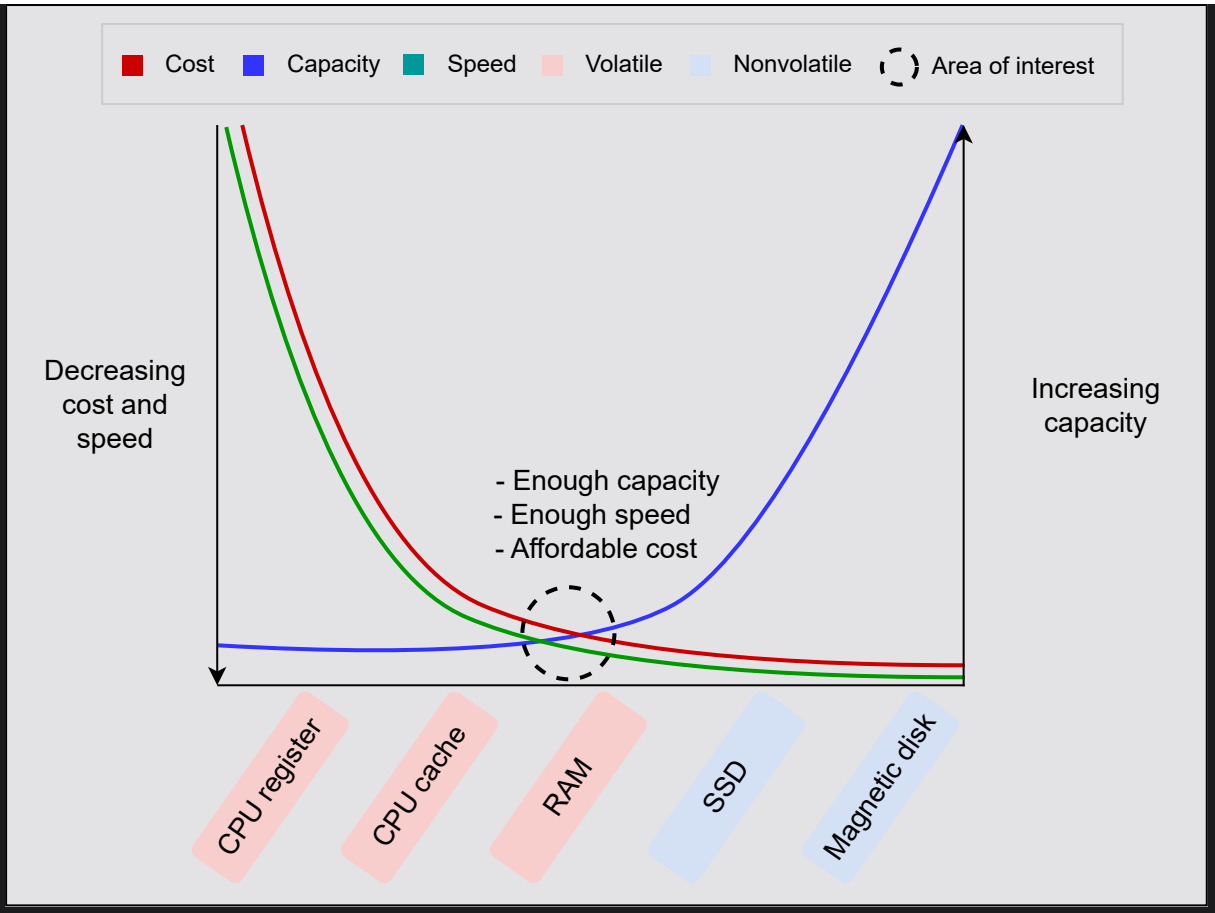
In such cases, a **cache** is added to the system to deal with performance deterioration. A cache is a temporary data storage that can serve data faster by keeping data entries in memory. Caches store only the most frequently accessed data. When a request reaches the serving host, it retrieves data from the cache (cache hit) and serves the user. However, if the data is unavailable in the cache (cache miss), the data will be queried from the database. Also, the cache is populated with the new value to avoid cache misses for the next time.




Service before using caching

1 of 2

A cache is a nonpersistent storage area used to keep repeatedly read and written data, which provides the end user with lower latency. Therefore, a cache must serve data from a storage component that is fast, has enough storage, and is affordable in terms of dollar cost as we scale the caching service. The following illustration highlights the suitability of RAM as the raw building block for caching:




An approximation that depicts how RAM is the optimal choice for serving cached data

We understand the need for a cache and suitable storage hardware, but what is distributed cache? Let's discuss this next.

## What is a distributed cache?

A **distributed cache** is a caching system where multiple cache servers coordinate to store frequently accessed data. Distributed caches are needed in environments where a single cache server isn't enough to store all the data. At the same time, it's scalable and guarantees a higher degree of availability.

Caches are generally small, frequently accessed, short-term storage with fast read time. Caches use the **locality of reference** principle.

Generally, distributed caches are beneficial in the following ways:

- They minimize user-perceived latency by precalculating results and storii frequently accessed data.
- They pre-generate expensive queries from the database.

- They store user session data temporarily.
- They serve data from temporary storage even if the data store is down temporarily.
- Finally, they reduce network costs by serving data from local resources.

## Why distributed cache?

When the size of data required in the cache increases, storing the entire data in one system is impractical. This is because of the following three reasons:

- It can be a potential single point of failure (SPOF).
- A system is designed in layers, and each layer should have its caching mechanism to ensure the decoupling of sensitive data from different layers.
- Caching at different locations helps reduce the serving latency at that layer.

In the table below, we describe how caching at different layers is performed through the use of various technologies. It's important to note that key-value store components are used in various layers.

### Caching at Different Layers of a System

| System Layer | Technology in Use | Usage |
|---|---|---|
| Web | HTTP cache headers, web accelerators, key-value store, CDNs, and so on | Accelerate retrieval of stat content, and manage ses |
| Application | Local cache and key-value data store | Accelerate application-l computations and data re |
| Database | Database cache, buffers, and key-value data store | Reduce data retrieval later I/O load from datab |

> Apart from the three system layers above, caching is also performed at DNS and client-side technologies like browsers or end-devices.

# How will we design distributed cache?

We'll divide the task of designing and reinforcing learning major concepts of distributed cache into five lessons:

1. **Background of Distributed Cache**: It's imperative to build the background knowledge necessary to make critical decisions when designing distributed caches. This lesson will revisit some basic but important concepts.
2. **High-level Design of a Distributed Cache**: We'll build a high-level design of a distributed cache in this lesson.
3. **Detailed Design of a Distributed Cache**: We'll identify some limitations of our high-level design and work toward a scalable, affordable, and performant solution.
4. **Evaluation of a Distributed Cache Design**: This lesson will evaluate our design for various non-functional requirements, such as scalability, consistency, availability, and so on.
5. **Memcached versus Redis**: We'll discuss well-known industrial solutions, namely Memcached and Redis. We'll also go through their details and compare their features to help us understand their potential use cases and how they relate to our design.

Let's begin by exploring the background of the distributed cache in the next lesson.