Notes on Caching Strategies for Read-Heavy Web Services

Sam Okafor

January 15, 2026

Abstract

We summarise three caching strategies commonly used in read-heavy web services — cache-aside, read-through, and write-behind — and give simple guidance for choosing between them based on consistency requirements and write volume.

Keywords

caching, web services, cache invalidation, read-through, write-behind

1 Introduction

Most web services read far more often than they write. When a database query becomes a bottleneck, an in-memory cache in front of it is usually the cheapest fix available, but the strategy used to keep that cache populated and correct has real consequences for both latency and data freshness.

2 Three strategies

2.1 Cache-aside

The application checks the cache first; on a miss, it reads from the database and populates the cache itself.

> Theorem 2.1 (Cache-aside invariant) Under cache-aside, a stale read is only possible in the window between a database write and the corresponding cache invalidation.

Theorem 2.1 is the reason cache-aside implementations must treat invalidation as part of the write path, not an afterthought.

2.2 Read-through

The cache itself is responsible for loading missing values from the database, so the application only ever talks to the cache.

2.3 Write-behind

Writes land in the cache first and are flushed to the database asynchronously. This gives the lowest write latency but risks losing recent writes on a cache failure.

3 Choosing a strategy

- If reads vastly outnumber writes and brief staleness is acceptable, cache-aside is simplest to reason about.
- If most access already goes through a single service layer, read-through centralises the loading logic.
- Write-behind should be reserved for cases where write latency dominates and some data loss on failure is tolerable.

4 Conclusion

None of these strategies is universally correct; the right choice follows from the read/write ratio and how much staleness or data loss a given endpoint can tolerate.