THM TECHNISCHE HOCHSCHULE MITTELHESSEN | CAMPUS FRIEDBERG | MND Mathematik, Naturwissenschaften und Datenverarbeitung

Microservices

Dokumentation

Caching verstehen

Schneller, effizienter, sicher

Verfasser der Arbeit

Kamal Badawi – 5236028

Betreuer

Torsten Steinmüller

Wintersemester 2025/26

26. Januar 2026

1. Einleitung

Caching (dt. Zwischenspeicherung) ist ein etabliertes Verfahren zur Performance-Optimierung, bei dem die Ergebnisse ressourcenintensiver Operationen temporär gespeichert werden. Dadurch können nachfolgende Anfragen mit identischen Parametern ohne erneute Berechnung oder Datenabfrage bedient werden. Das Kernziel ist die Reduktion von Latenz und die Entlastung primärer Systemkomponenten.

Eine anschauliche Analogie ist die Pufferlagerung in der Logistik: Anstatt jeden Kundenauftrag individuell und direkt vom Zentrallager zu bearbeiten, werden häufig nachgefragte Artikel in dezentralen Regionallagern vorgehalten. Dies minimiert die Auslieferungszeit und entlastet die zentrale Infrastruktur.

2. Ziele des Cachings

Die Implementierung von Caching-Systemen verfolgt zwei primäre Ziele:

- **Latenzreduktion:** Bereitstellung von Daten und Inhalten mit minimaler Verzögerung für den Endnutzer.
- **Ressourcenoptimierung:** Entlastung von Backend-Systemen (z. B. Datenbanken, Applikationsserver) durch Reduktion redundanter Operationen, was Skalierbarkeit und Kosteneffizienz steigert.

3. Architekturebenen des Cachings

In modernen Webarchitekturen wird Caching typischerweise auf drei hierarchischen Ebenen implementiert:

3.1. Client-seitiges Caching (Browser-Cache)

- **Speicherort:** Lokal auf dem Endgerät des Nutzers (Webbrowser).
- **Anwendungsfall:** Zwischenspeicherung statischer Ressourcen wie Stylesheets (CSS), Skripte (JavaScript), Bilder und Schriftarten.
- **Steuerung:** Serverseitig via HTTP-Header, z. B. Cache-Control: max-age=60. Dies instruiert den Browser, die Ressource für 60 Sekunden aus dem lokalen Cache zu bedienen, bevor eine Validierung beim Server erfolgt.

3.2. Proxy- bzw. CDN-Caching (Content Delivery Network)

- **Speicherort:** Auf geografisch verteilten Edge-Servern eines CDN-Anbieters.

- **Anwendungsfall:** Auslieferung öffentlicher, statischer oder semi-statischer Inhalte (z. B. Medien, Software-Downloads) an ein globales Publikum.
- **Vorteil:** Signifikante Reduktion der Netzwerklatenz (Round-Trip Time) durch physische Nähe zum Nutzer sowie massive Entlastung des Origin-Servers.

3.3. Server-seitiges Caching (Application-/Backend-Caching)

- **Speicherort:** Im Arbeitsspeicher des Applikationsservers oder in spezialisierten In-Memory-Datenspeichern wie Redis oder Memcached.
- **Anwendungsfall:** Zwischenspeicherung der Ergebnisse komplexer oder häufiger Berechnungen, z. B. aufwändige Datenbankabfragen oder gerenderte Templates.
- **Vorteil & Steuerung:** Maximale Kontrolle durch die Anwendungslogik. Beispielsweise können Cache-Eintrücke bei Datenänderungen gezielt invalidiert werden. Frameworks bieten hierfür dedizierte Mechanismen (z. B. @functools.lru_cache in Python).

4. Herausforderungen und Best Practices

Trotz seiner Vorteile erfordert Caching eine sorgfältige Strategie, um zentrale Risiken zu vermeiden.

4.1. Konsistenzprobleme (Stale Data)

- **Problem:** Der Cache enthält veraltete Daten, während die Originalsource bereits aktualisierte Werte bereithält.
- **Lösungsansätze:**
  - Time-to-Live (TTL): Jeder Cache-Eintrag erhält eine feste Gültigkeitsdauer.
  - Ereignisgesteuerte Invalidierung: Der Cache-Eintrag wird explizit gelöscht oder aktualisiert, sobald die Quelldaten geändert werden (z. B. via Publish/Subscribe-Muster).

4.2. Datenschutz und Sicherheit (Data Leakage)

- **Problem:** Unsachgemäße Konfiguration kann dazu führen, dass personenbezogene oder session-spezifische Daten (z. B. Nutzerprofile, Warenkörbe) in einem öffentlich zugänglichen Cache (z. B. CDN) gespeichert und so anderen Nutzern zugänglich werden.

- **Lösungsansätze:**
  - Verwendung des Cache-Control: private Headers, um die Speicherung ausschließlich im privaten Browser-Cache zu erlauben.
  - Einsatz des Vary Headers (z. B. Vary: Cookie, Authorization), um Cache-Versionen abhängig von Authentifizierungs- oder Session-Headern zu unterscheiden.

5. Fazit

Caching stellt ein fundamentales und leistungsstarkes Instrument zur Optimierung moderner IT-Systeme dar. Sein effektiver Einsatz steigert die Benutzererfahrung durch reduzierte Ladezeiten und erhöht gleichzeitig die Robustheit und Skalierbarkeit der Infrastruktur. Der erfolgreiche Betrieb setzt jedoch ein durchdachtes Konzept zur Gültigkeitssteuerung (Invalidierung) und zur Absicherung sensibler Daten voraus. Bei korrekter Implementierung wird Caching damit zu einem zentralen Enabler für performante, effiziente und zuverlässige Anwendungen.