Day 9: Caching Mechanisms & Multi-threading

This day focuses on **performance engineering**: caching layers (L1/L2/L3) and **concurrency models** in Node.js and Python. These topics are critical for scalable backend and system design interviews.

> Assumption continued: **No Docker usage** (native/local services only).

---

1. Multi-Level Caching Strategy Overview

Cache Layers

Level	Type	Scope	Latency	Example
L1	In-memory	Process	~ns–µs	Python dict, Node Map
L2	Distributed	Network	~ms	Redis
L3	CDN	Global	~10–50 ms	Cloudflare

Request flow:

```
Client → CDN (L3)
       → App → L1 Cache
             → L2 Cache (Redis)
                   → Database
```

---

2. L1 Cache – In-Memory with TTL & Eviction

Design Goals

- Fastest access
- Per-process
- TTL-based expiration
- Eviction policy (LRU)

Python L1 Cache (TTL + LRU)

```python
import time
from collections import OrderedDict

class L1Cache:
```

```
    def __init__(self, capacity=100, ttl=60):
        self.cache = OrderedDict()
        self.capacity = capacity
        self.ttl = ttl

    def get(self, key):
        if key not in self.cache:
            return None
        value, timestamp = self.cache.pop(key)
        if time.time() - timestamp > self.ttl:
            return None
        self.cache[key] = (value, timestamp)
        return value

    def set(self, key, value):
        if key in self.cache:
            self.cache.pop(key)
        elif len(self.cache) >= self.capacity:
            self.cache.popitem(last=False)  # LRU eviction
        self.cache[key] = (value, time.time())
```

✔️Checklist: L1 cache with TTL + eviction

---

3. L2 Cache – Redis (Cache-Aside Pattern)

Cache-Aside Pattern

1. Check cache
2. Cache miss → DB
3. Store result in cache
4. Return response

Redis Setup (Local Service)

```
redis-server
```

Python Redis Integration

```
pip install redis
```

```
import redis
import json
```

```python
r = redis.Redis(host="localhost", port=6379, decode_responses=True)

def get_user(user_id):
    cache_key = f"user:{user_id}"
    cached = r.get(cache_key)

    if cached:
        return json.loads(cached)

    # Simulated DB fetch
    user = {"id": user_id, "name": "Ashu"}

    r.setex(cache_key, 300, json.dumps(user))
    return user
```

✔️Checklist: Redis cache-aside implemented

---

4. Cache Invalidation Strategies

Common Patterns

Strategy	Use Case
TTL-based	Frequently changing data
Write-through	Strong consistency
Write-behind	High throughput
Explicit delete	User updates

Example (Explicit Invalidation)

```python
r.delete(f"user:{user_id}")
```

Rule of thumb:

> *Cache invalidation is harder than caching — favor TTL + delete-on-update.*

---

5. L3 Cache – CDN (Cloudflare)

What CDN Caches

- Static assets (JS, CSS, images)

- API GET responses (selectively)

Cloudflare Cache Rules

Example rules: - Cache `GET /api/public/*` for 10 minutes - Bypass cache for authenticated routes

HTTP Headers for CDN Caching

```
Cache-Control: public, max-age=600
ETag: "v1-user-list"
```

✔️Checklist: CDN caching rules configured

---

6. Node.js – Worker Threads (CPU-Intensive Tasks)

Why Worker Threads

- Node.js event loop blocks on CPU-heavy tasks
- Workers offload computation

Worker File (`worker.js`)

```
const { parentPort } = require('worker_threads');

parentPort.on('message', (data) => {
  let result = 0;
  for (let i = 0; i < 1e9; i++) result += i;
  parentPort.postMessage(result);
});
```

Main Thread

```
const { Worker } = require('worker_threads');

const worker = new Worker('./worker.js');
worker.postMessage('start');
worker.on('message', result => console.log(result));
```

✔️Checklist: Worker threads processing background jobs

---

7. Python – Multiprocessing

When to Use

- CPU-bound tasks
- Bypass GIL

```
from multiprocessing import Pool, cpu_count

def cpu_task(n):
    return sum(i*i for i in range(n))

with Pool(cpu_count()) as p:
    results = p.map(cpu_task, [10_000_000] * 4)
```

---

8. Python – AsyncIO (I/O Bound)

When to Use

- Network calls
- DB queries
- File I/O

```
import asyncio

async def fetch_data(i):
    await asyncio.sleep(1)
    return i

async def main():
    results = await asyncio.gather(*(fetch_data(i) for i in range(5)))
    print(results)

asyncio.run(main())
```

---

9. Cache Performance Metrics

Metrics to Track

- Cache hit rate
- Miss rate
- Latency reduction

Example Metrics

Layer	Hit Rate	Avg Latency
L1	92%	0.2 ms
L2 (Redis)	85%	1.5 ms
DB	—	15 ms

Hit Rate Formula:

```
hit_rate = hits / (hits + misses)
```

✔️Checklist: Cache hit rate > 80%

Day 9 Completion Status

- ☑️L1 cache with TTL & eviction
- ☑️Redis cache-aside pattern
- ☑️CDN caching rules (Cloudflare)
- ☑️Node.js worker threads
- ☑️Python multiprocessing & asyncio
- ☑️Cache performance metrics documented

Key Interview Insight

> Performance ≠ faster code Performance = **caching + concurrency + correct architecture**

Next steps: **Day 10 – Observability & Monitoring** (Logs, metrics, tracing, Prometheus, OpenTelemetry)

Or integrate **Redis + L1 cache into your existing FastAPI backend**.