Architecture of Large-Scale Systems

By Dr. Taghinezhad

Mail:
a0taghinezhad@gmail.com

CHAPTER 2 Two Mistakes High—Having Room to Recover from Mistakes

The Importance of Redundancy and Availability

- a MySQL database backup replica being used for experimentation, leading to a failure during a primary database outage.
  - Problem, Can backup replica be used for production, when it is experimented ?
    - No, because its setting is changed and it is not longer reliable
- **Key Lesson:** Backup systems must be treated with the same rigor as primary systems to ensure availability.

- **Takeaway:** Redundant systems are not just backups; they are critical components in maintaining high availability

Two Mistakes High: Philosophy of Recovery

- **Concept:** "Two mistakes high" from radio-controlled planes.
- **Key Idea:** Always keep enough "altitude" (resources) to recover from two independent mistakes.
- **Application:** In highly available systems, plan for multiple failures and ensure recovery from any combination of mistakes.

Scenario #1: Node Failure

- **Initial Setup:** Service designed to handle 1,000 req/sec with 4 nodes (300 req/sec each).
- Question: How many nodes do you need to handle your traffic demands? Some basic math should come up with a good answer:
  - $number\_of\_nodes\_needed = \lceil \frac{number\_of\_requests}{requests\_per\_node} \rceil$
  - can you handle the expected traffic, and because you have four nodes, you can handle the loss of a node?

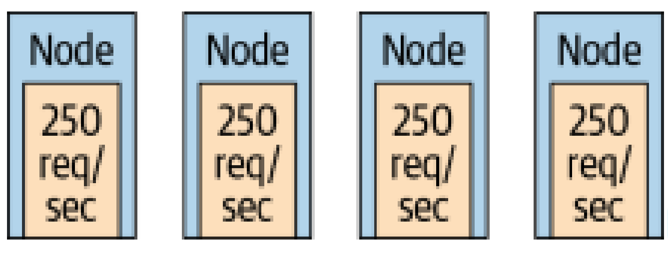

Figure 2-1. Four nodes, 250 req/sec each

Scenario #1: One Node Failure

- **Failure Situation:** One node fails; remaining nodes overloaded, leading to service degradation (Figure 2-2).
  - requests_per_node = 1,000 req/sec / 3 nodes = 333 req/sec/node
  - That's 333 req/sec per node, which is well above your 300 req/sec node limit (see Figure 2-2).

- **Solution:** Add a 5th node to ensure capacity even after one failure (Figure 2-3).

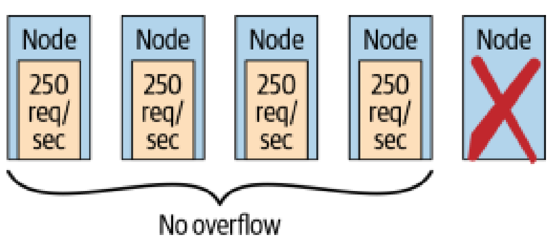

Figure 2-3. Five nodes; one failure can still be handled

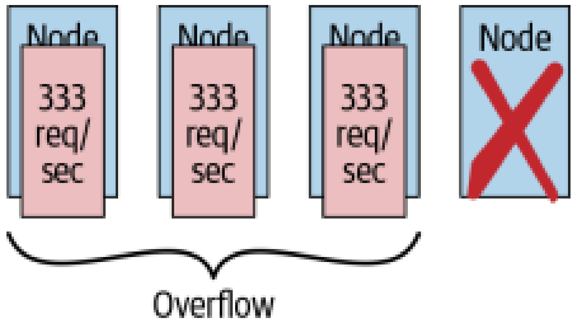

Figure 2-2. Four nodes; one failure causes overflow

Scenario #2: Rolling Upgrades and Node Failures

- **Upgrade Plan:** Rolling deploy with 5 nodes ensures availability during upgrades.
- **Risk:** A node failure during an upgrade leaves only 3 nodes handling traffic, leading to an outage.
- **Lesson:** Ensure redundancy covers both routine maintenance and unexpected failures.

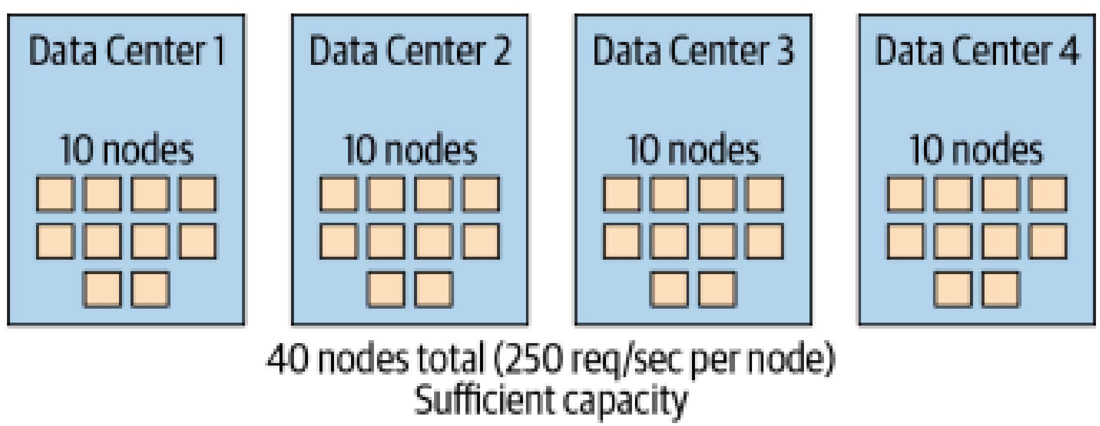

Figure 2-4. Four data centers, 40 nodes, sufficient capacity to handle load

Scenario #2: Rolling Upgrades and Node Failures

- Suppose that you have a service whose average traffic is 1,000 req/sec.
- let’s assume that a single node in your service can handle 300 req/sec.
  - Four node is enough to handle expected traffic
- You want to do a software upgrade while running your service nodes.
  - A rolling deploy (upgrade nodes one by one to keep operational reset when one is upgrading).
  - How many nodes is needed?
  - Five Nodes
    - This system can tolerate single node failure and support rolling deploy updates
  - Six Nodes, can handle multimode failure

Scenario #3: Data Center Resiliency

- **Setup:** Service requires to handle 10,000 req/sec,
  - It would need 34 nodes without considering redundancy for failures.
  - let’s uses 40 nodes across four data centers so that we have even more redundancy and fault tolorance.
  - Are we resilient?

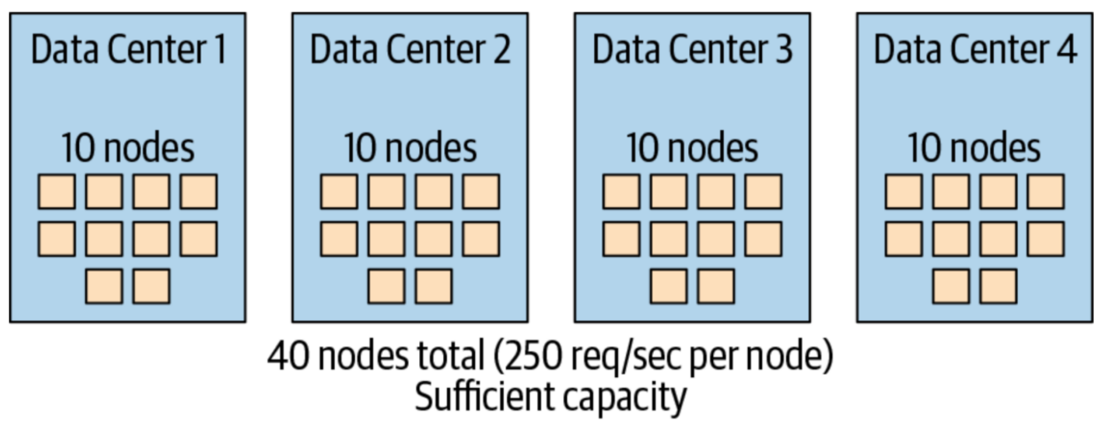

Figure 2-4. Four data centers, 40 nodes, sufficient capacity to handle load

Scenario #3: Data Center Resiliency (Cont.)

- **Risk:** One data center outage leads to overloading the remaining nodes (Figure 2-5).

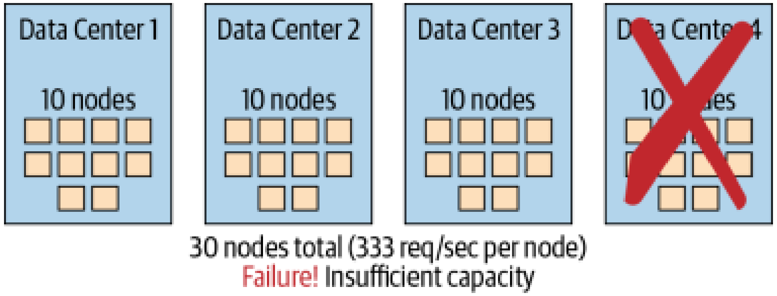

Figure 2-5. Four data centers, one failed, 30 nodes, insufficient capacity to handle load

- How many servers we need to handle lose of a datacenter?
  - $nodes\_per\_data\_center = \lceil \min\_number\_of\_servers / (number\_of\_data\_centers - 1) \rceil$
- **Solution:** To maintain capacity with a data center outage, 48 nodes are needed.
  - $nodes\_per\_data\_center = \left\lceil \frac{34}{(4-1)} \right\rceil$ =12 server/data_center

  - How many nodes will it be?
    - total_nodes = nodes_per_data_center × 4 = 48 nodes

Scenario #4: Hidden Shared Failure Types

- Sometimes seemingly independent problem scenarios can actually be dependent, meaning they might fail together
- **Example:** Your service needs four nodes, but you've wisely prepared with six nodes—enough to handle a single node failure and an upgrade in progress.
  - Six nodes sharing the same rack and power supply all fail simultaneously.
- **Key Point:** Ensure physical and infrastructure-level separation to prevent cascading failures.

Scenario #5: Failure Loops

- A failure loop occurs when a problem prevents you from fixing it without causing a worse issue
  - **Example:** Imagine having a backup generator stored in your garage, but the only way to access the garage is through an electric-powered door that doesn't work during a power outage. Similarly, in the world of services, dependencies between failures and solutions can impact availability.
- **Lesson:** Ensure that backup systems can be activated even during failures, avoiding failure loops.

Managing Your Applications

- **Key Principles to manage your applications:**
  - **"Fly Two Mistakes High"**
    - Look beyond surface failure modes.
    - Consider dependent failure layers.
    - Ensure recovery mechanisms work during failures.
  - **Don't Ignore Problems**
    - Persistent issues affect availability plans.
    - Backup systems matter—treat them seriously.
  - **Production Is Production**
    - Everything in production matters.
    - Backup databases are mission-critical too.
  - **Layered Failures Are Tricky**
    - Identifying dependencies isn't obvious.
    - Invest time in understanding and resolving.

Case Study: Space Shuttle Redundancy

- The Space Shuttle software system was one of the first large-scale applications to implement extreme redundancy and failure management.
  - Primary system: 5 computers (4 identical running the same software, 1 independent).
  - Main process on all computes during critical parts:
    - •4 computers received the same data and performed the same calculations.
      - •If one computer differed, it was voted out and shut down as it was uncorrect.(winners rule, loses terminate)
      - •The shuttle could operate with 3 computers and land safely with 2.

Case Study: Space Shuttle Redundancy (Cont.)

- what would happen if the four computers couldn’t agree? This could happen if there were multiple failures and multiple computers had been shut down.
- Solution:
  - **System Setup:** 5-computer redundancy system on the Space Shuttle.
    - **4 main computers** with identical software that vote on outputs.
    - **1 independent computer** with simpler software to resolve disputes.

  - **Outcome:** 30 years of successful missions with no life-threatening software failures.

Case Study: Space Shuttle Redundancy

- **System Setup:** 5-computer redundancy system on the Space Shuttle.
  - **4 main computers** with identical software that vote on outputs.
  - **1 independent computer** with simpler software to resolve disputes.
- **Outcome:** 30 years of successful missions with no life-threatening software failures.

- End of Chapter 2