High Availability

2023

This presentation is based on

- Chapter 2: “Ensuring High Availability for Your Enterprise Web Applications”

Book: “Architecting High Performing, Scalable and Available Enterprise Web Applications” by: Shailesh Kumar Shivakumar, 2015

1. Introduction

Availability

- High availability is the ability of a system to be continuously available to users without any loss of service.
- To achieve five nines (99.999%) availability, the maximum allowed downtime is 300 s per year. 5 minutes.

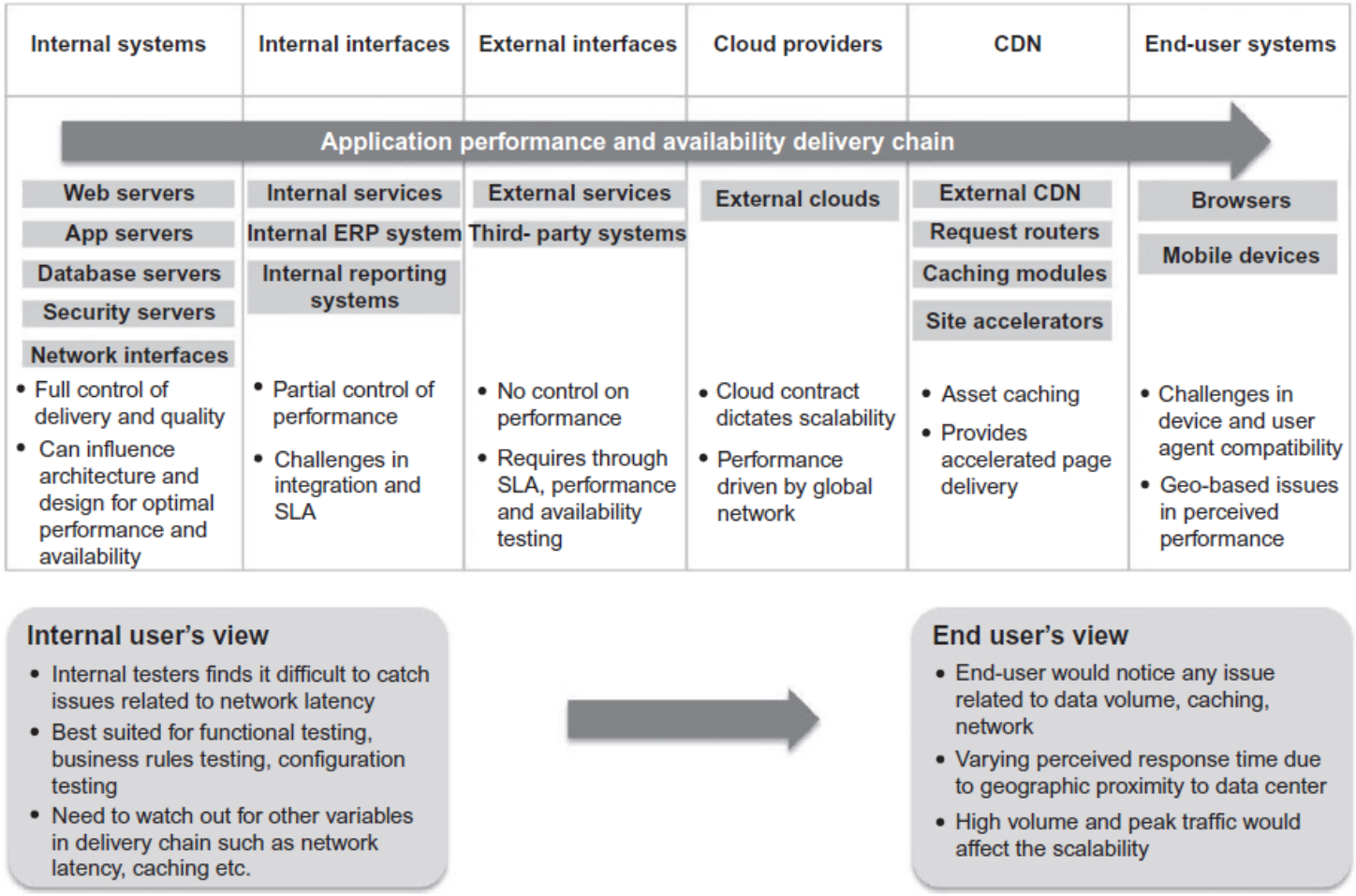
Example

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

2. Key principles

- Satisfy SLA
- Support failover and fallback
- Proactive monitoring
- Manage planned and unplanned downtime
- Handle all possible point of failure
- Robust infraTI by sufficient redundancy

Enterprise application availability chain



Availability establishment process

- key availability metric parameters:
  - System availability: total uptime per given time period.
    - Ej: 99.999%
  - Mean Time to Recover (MTTR): average time for system to recover from a failure.
    - MTTR = (Total downtime) / (Total number of system failures)
  - Mean Time Between Failures (MTBF): average time between two system failures.
    - MTBF = (Total uptime) / (Total number of system failures)
  - Service availability: amount of time the business services are available for a given time period.

Establish availability design criteria.

- Design for failure (ex. Netflix)
- Design for handling downtime and recovery.
- Design for continuous operations and business continuity.

Challenges to high availability

- Hardware-related challenges
  - Non-scalable infrastructure
  - Network bandwidth challenges:
  - Hardware issues:
  - Existence of single points-of-failure (SPOF):
  - Existence of non-scalable choking points (CP):

Challenges to high availability

- Software-related challenges
  - Application design and coding issues
  - Upstream and enterprise interface challenges
  - Security issues
  - Improper or nonexistent caching strategy
  - Absence of fool-proof availability test cases

High availability architecture patterns

- Failover
- Failback
- Replication
- Redundancy
- Virtualization
- Continuous maintenance:
  - Corrective maintenance
  - Preventive maintenance
  - Perfective maintenance

Software high availability patterns

- Graceful and step-wise functionality degradation pattern
- Asynchronous and services-based integration with external interfaces:
- Stateless and lightweight application components:
- Continuous incremental code and data replication:
- Availability trade-off using the CAP theorem (CAP: Consistency, Availibility, Partition Tolerance)

High availability best practices

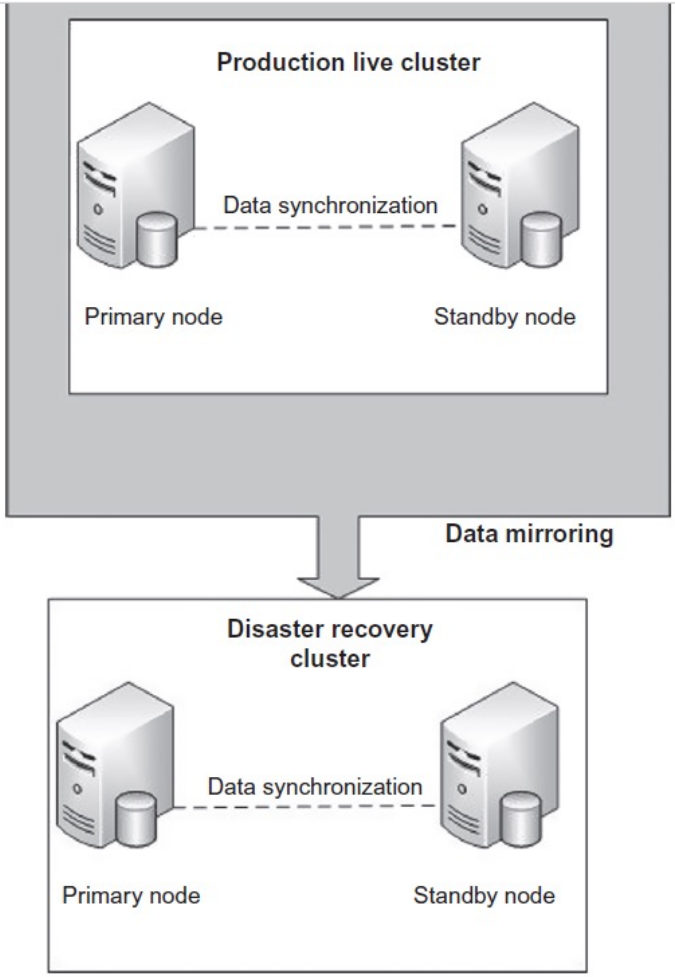
- Hardware-related best practices
  - proactive monitoring and alerting infrastructure
  - hardware redundancy
  - disaster recovery
- Software-related best practices
  - architecture simple
  - Design modular software components
  - caching strategy
  - automation for maintenance activities

High availability for storage

- Storage availability through RAID
- Storage virtualization
- Storage availability through NAS and SAN

High availability database

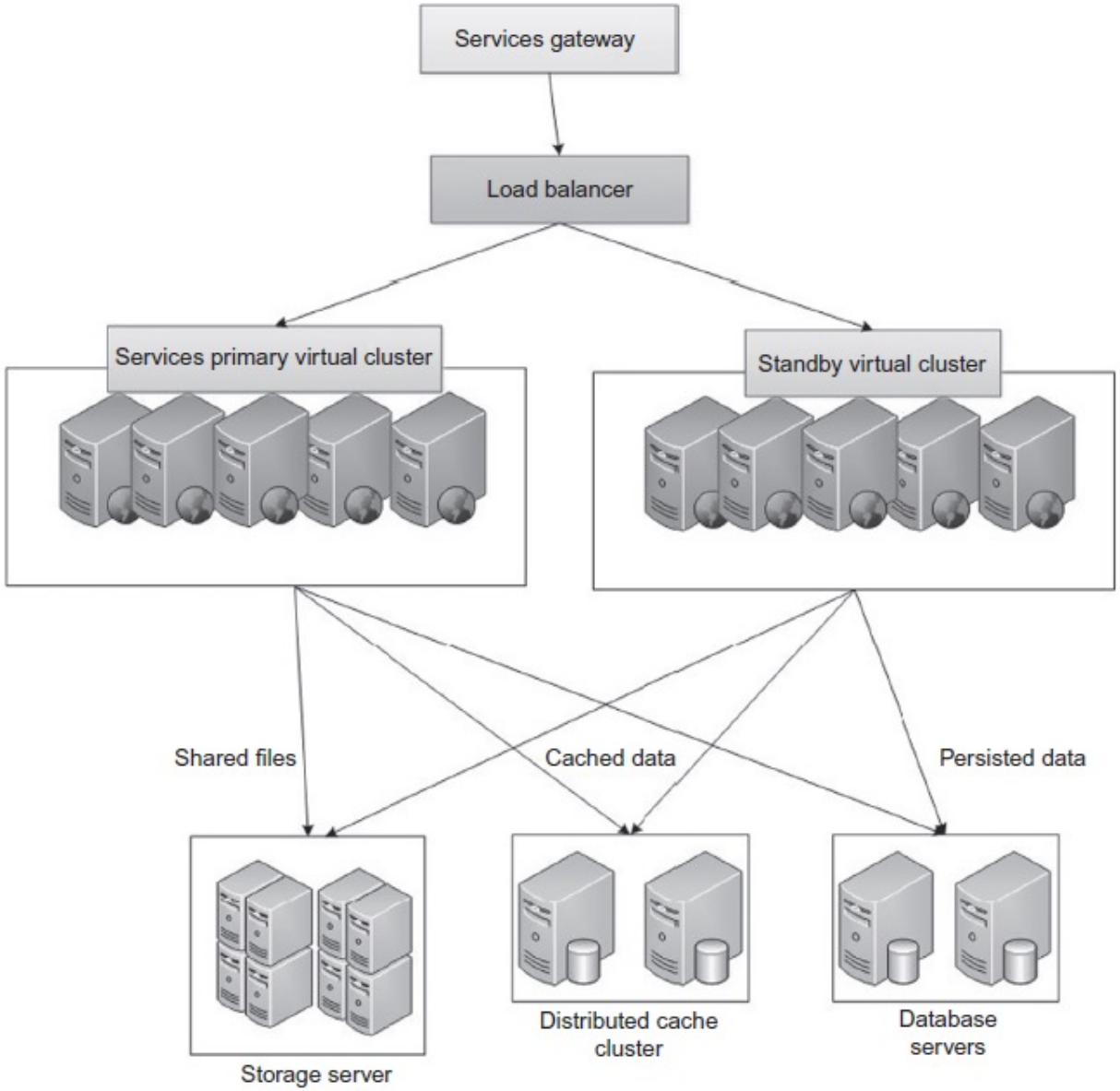
- database clustered configuration



Network availability

- Redundancy links
- monitoring of network
- failover features
- network-level fault detection, self-recovery, and graceful restart

Cluster-based high availability architecture for hosted services



Availability tactics

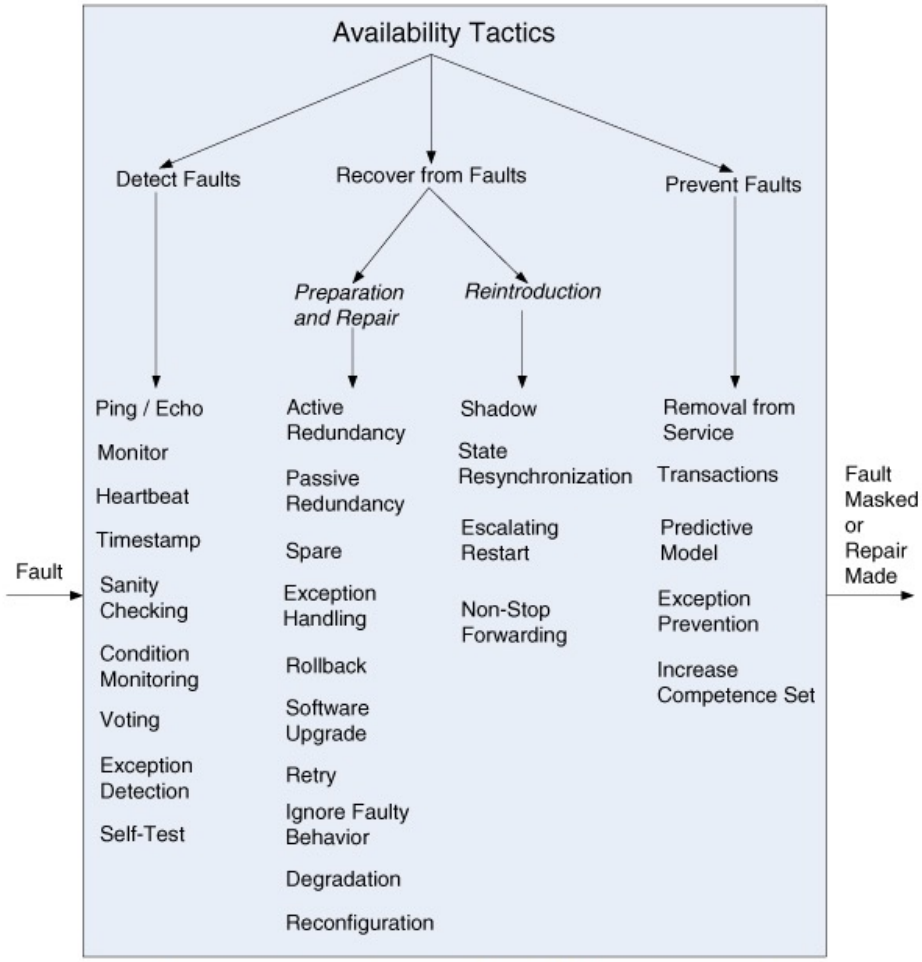


Figure 5.5. Availability tactics

3. Availability through fault tolerance and failover

Fault tolerance

- Fault tolerance is an attribute of the system wherein the system can survive any component faults (or failures) through various failover techniques.

Software fault tolerance

- Faults due design-time and runtime issues
- Exception handling routine vs foult-tolerant routine (stop or continue) –
  - FT -> detect and handle the faults.
- Application code analysis and fault handlers.
- Recovery using checkpoint and rollback.
  * Mainly used for databases systems and operating systems
  - In application: ex: store session
- N-versión software
  - Each versión of a component is designed and implemented in a different way.
- Fault handling through fallback
  - Fallback procedures to take alternate course of action, with degradation of functionality.

Software fault tolerance

Summary:

1. Adding redundant software components and creating n-version software modules

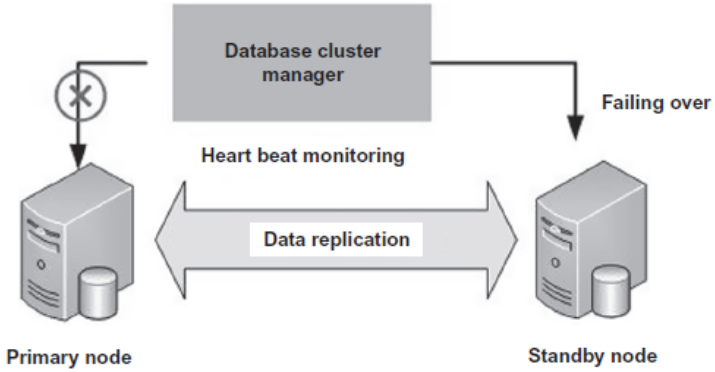
2. Designing fallback routines, which sense the component failure and execute an alternate course of action with available components.

Hardware fault tolerance - 1

- Hardware component redundancy
  - N + 1 or N + M (N main component, 1 or M redundant)
  - Levels:
    - Node level
    - Cluster Level
    - Site Level (disater recovery and business continuity)
- Replication
  - Data and configuration in redundant nodes (support failover strategy)
  - Types:
    - Synchronous and Asynchronous
- Fault detection and isolation

Hardware fault tolerance - 2

- Failover
  - Once a fault is detetect, the requests are failed over a healty nodes.
  - Used by Load Balancers and Cluster Mgt
  - Data consistence???



Database availability

- Database storage (discos en RAID)
- Database cluster (me permite escalar)
- Indexing
- Data replication (tolerancia a fallos)
- Partioning (escalar)
- Data snapshots (hacer recuperación) por replicación
- Backup and recovery (lo mínimo que debería tener)