System Scalability Cheat Sheet

by saran sudha via cheatography.com/202593/cs/43005/

System scalability

System scalability refers to the ability of a computer system, network, or software application to handle an increasing amount of work, such as additional users, higher data volumes, or more complex processing, while maintaining acceptable performance levels. Scalability is a crucial aspect of system design, particularly in environments where growth, fluctuations in demand, or changes in requirements are expected. Here are some key aspects of system scalability

Scalability vs Performance

Scalability refers to the ability of a system to handle an increasing workload or user base while maintaining acceptable performance levels. It focuses on the system's ability to scale up or scale down in response to increased/decreased demand.	Performance refers to how well a system performs a specific task or operation, often measured in terms of speed, response time, throughput, or latency.
Unlike performance, scalability should always be measured under variable load	Performance optimization aims to maximize the speed, efficiency, and responsiveness of the system under a fixed workload

Scalability principle

Decentralization
Independence
Modularity

Decentralization

- Monolith is an anti-pattern for scalability.
- Involves distributing tasks, responsibilities, and resources across multiple nodes.

Independence

Independence refers to the ability of system components to operate autonomously and independently of each other.

Modularity

- Scalable architectures often start with modularity
- Allows components to be independently developed, deployed, and scaled
- Each module performs a specific function
- can be replicated or distributed as needed

Scalability Types:

- **Vertical scaling**
- **Horizontal scaling**

Vertical scaling

Vertical scaling (cont)

- Single Point of Failure
- Cost-Effectiveness

Horizontal scaling

- Adding more servers or machines or nodes
- Additional resources work in parallel to distribute the load
- Allows the system to handle increased traffic

Benefits

- Unlimited scalability
- Better Performance
- High Availability
- Cost-Effectiveness

Challenges

- Hard to achieve
- Data Consistency

Horizontal scalability

- **Replication**
- **Services**
- **Caching**
- **Asynchronous process**
- **Partitioning**

Replication applications

Stateless:

Code replication

Stateful:

Code & Data replication

Stateless replication (cont)

- Lower communication overhead

Challenges

- Session Management
- Consistency and Synchronization
- Monitoring and Load Balancing

Stateful replication

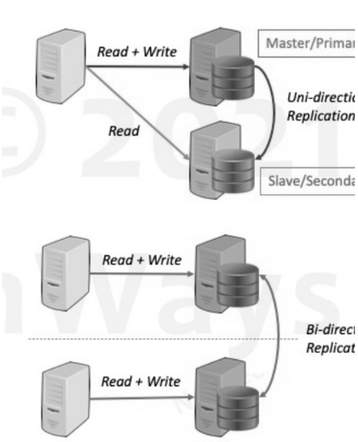
- Maintains session data or other infos of users/sessions
- All replicas maintain synchronized state
- Includes database replication across instances

Challenges

- Session Management
- Data Consistency
- Synchronization Overhead
- Failure Handling

Database replication

Database replication



Caching

- Alleviates the load on backend resources
- Improves system performance
- Reduces latency
- cached data across multiple cache servers or instances
- support horizontal scaling

Partitioning

Increase the resources (CPU, RAM, etc.) of a single server.

Enhancing the capabilities of existing hardware

Benefits

Easier to implement

Easier to manage

Challenges

Limited scalability

Hardware constraints

Stateless replication

Each request is processed independently

Each replica can operate autonomously

Does not maintain any state or session data

Components can be replicated across multiple servers or instances

Benefits

Higher data availability

Reduced server load

More reliable data

Better protection

Lower latency

Better application performance

Challenges

Inconsistent data

Lost data

Database replication types

Master-Slave Replication

Master-Master Replication

not going to discuss about in details about types as most of are aware of it

Splits a large dataset or workload into smaller ones

Distributed across multiple servers or nodes

Benefits

Handles increased data volume

Handles higher user concurrency

Handles workload demand more effectively

Types:

Vertical Partitioning

Horizontal Partitioning

Vertical partitioning

Splits a large dataset into smaller partitions based on the functionality

Completely decouples services and databases for higher scalability

Benefits

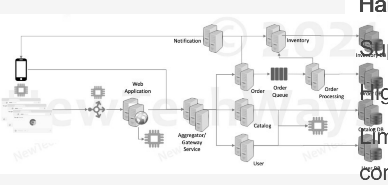
Vertical partitioning (cont)

- Efficient Resource Utilization
- Flexible Scalability Options
- Tailored Data Management

Challanges

- Data Distribution and Access Patterns
- Limitations in number of partitioning
- Data Locality and Access Patterns

Vertical partitioning



In the image DB are splitted into Inventory, Order, Catalog and User

Horizontal partitioning

- Splits a large dataset horizontally into smaller partitions. Rows are divided into smaller sets, and each set is stored separately.
- Partition based on ranges of data, hash values, or other partitioning keys.
- Distributes data and workload across multiple servers
- Handle increased data volume

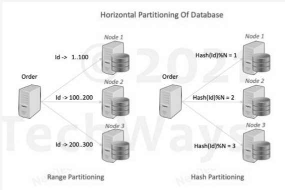
Benefits

- Increased Throughput
- Enhanced Scalability
- Partitioning Strategies

Challanges

- Data Distribution
- Data Consistency and Integrity
- Partition Management and Maintenance

Horizontal partitioning



Load Balaner

- Distributes incoming traffic
- Ensures optimal performance and reliability

Types

Hardware Load Balance

- Supports L4* and L7*
- Higher cost
- Limited flexibility and agility compared

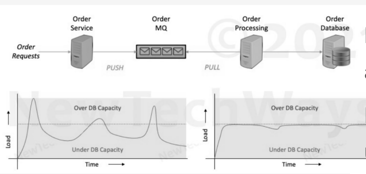
Software Load Balancer

- Supports only L7*
- Lower cost compared to HLB
- Greater flexibility and agility in deployment and management.

L4 - Transport Layer of the OSI includes UDP,TCP, SCTP

L7 - Application Layer of the OSI which is highest layer includes HTTP, HTTPS, SFTP etc

Asynchronous services



Asynchronous services

- Decoupling tasks or processes from synchronous execution
- Improves system responsiveness, resource utilization, and fault tolerance

DNS as load balancer

- Involves leveraging DNS to distribute incoming client requests across multiple backend servers or resources.
- Returns single IP in round robin fashion
- can be configured along with health checks

Drawback

- Indefinite caching
- Low or zero TTLs
- Very high load on DNS

SAGA Pattern

- Atomic Transactions
- Asynchronous Communication
- Horizontal Scaling
- Fault Tolerance
- Dynamic Adaptation
- Isolation of Operations

Micro service architecture style

Shared Nothing architecture

- Services developed and deployed independently
- Achieved through vertical partitioning

Vertical/Domain partitioning

- Independent schema/database
- Loosely coupled services interface(REST interfaces)
- No reusable libraries except utilities

Challenges

- Duplicate codebase
- Transaction failures
- Transaction rollbacks

Discovery services

- Manage service-to-service communication in a distributed system
- Facilitate the dynamic discovery
- Continuously monitors the health and availability of registered microservices.

Benefits

- Service Registration and Discovery
- Dynamic Load Balancing
- Fault Tolerance and Failover
- Service Scaling and Elasticity

Conclusion

- Scalable systems are decentralized and functions independently

To make a system scalable

- Cache frequently read and rarely mutating data
- Asynchronous or Event driven process
- Vertical partitioning of functionality into independent, stateless, replicated services
- Partitioning and replication for extreme scalability

Scalable systems infra

- Load balancers - Hardware based & Software based
- Discovery services for service discovery and health checks
- DNS as load balancer

Microservices

- Fully vertically partitioned services and databases leads to eventual consistency