KAPITEL 5 / *CHAPTER 5* [5]
RESILIENT WEB-APPLICATION SCALING THROUGH INTEGRATED DATABASE ELASTICITY

DOI: 10.30890/2709-2313.2024-28-00-002

Introduction

This work has focused on the optimization and scaling of web applications, particularly those that utilize databases. The work covered various deployment environments, including physical servers, dedicated servers, virtual servers, cloud systems, and Service Mesh. It highlighted the importance of scalability and fault tolerance, with an emphasis on autoscaling to improve system responsiveness and reduce manual configuration.

The work also delved into database management, discussing the challenges of maintaining stateful databases and ensuring data consistency. The "Sharding + Master-Slave Replication" method was identified as an effective strategy for database scaling, allowing for potentially unlimited growth without sacrificing access speed or fault tolerance.

Service Mesh, particularly Kubernetes, was recommended as the best solution for scaling web applications due to its flexibility, cost-effectiveness, and alignment with modern practices like containerization and Infrastructure as Code (IoC). The conversation concluded that Service Mesh provides the necessary infrastructure to support long-term user needs and is equipped with autoscaling as a core feature, making it the most advanced option for deploying and managing web applications.

5.1. Current condition

The history of web resources traces back to the 1990s with the publication of the first hypertext document on the internet. This marked the beginning of a rapid

[5]***Authors:*** *Korobeinikova Tetiana Ivanivna*

evolution; from simple HTML documents viewed in browsers to the development of pages with dynamic elements. Today, we witness multiple generations of the World Wide Web, featuring intricate distributed systems capable of sophisticated calculations and data processing. These systems, along with additional subsystems, manage complex software and server operations, all aimed at efficiently handling user resources. As the world moves faster, the demand for instant information access has grown, leaving companies that failed to adapt losing their influence in the IT realm [1-3].

Internet usage and device connectivity have surged, with daily increases in penetration rates. It's become unimaginable to operate electronics without network access. This growth trajectory has compelled companies to prioritize product speed and reliability, necessitating the hiring of top-tier professionals from the inception of product development. Modern services extend beyond the capabilities of HTML and CSS, as these technologies alone cannot foster dynamic applications. The focus has shifted to servers, which are often composed of numerous subsystems, applications, and microservices. A typical server setup includes at least one application, a web server for access management, and a database for storing user and system data. Some configurations may also incorporate queue servers, cache servers, and various other systems [4-7].

The success of IT projects is gauged by user numbers, which directly influence revenue. It's common to observe a multitude of users accessing a single application simultaneously. However, many projects are ill-prepared for organic growth, leading to system failures under heavy loads. Overload issues manifest as reduced resource performance, increased errors, equipment malfunctions, and intermittent unavailability of certain components, all of which diminish uptime—a critical metric that, if compromised, can drive users to competitors [8].

A dichotomy exists between the continuous growth of users and the need for fault tolerance in web application development and maintenance. This tension renders the issue of application overload both pertinent and ripe for exploration in the scientific domain.

5.2. Resilience in Overload: Ensuring Stability and Continuity in Web Applications

When an application is overwhelmed with traffic, it becomes prone to instability, and the malfunction of any part can lead to prolonged recovery times or even result in data corruption or loss. An overloaded system often remains offline until a technician completes the restoration process. Fault tolerance refers to a system's resilience against unexpected software/hardware malfunctions or issues within the OSI network layers, ensuring minimal downtime. A fault-tolerant system is designed to remain operational despite adverse conditions, with the capability for manual or automatic recovery of non-operational nodes. In contrast, a system lacking fault tolerance cannot autonomously repair its impaired components following an anomaly [10-12].

Recovery mechanisms in fault-tolerant systems typically involve either a 'run' or a 'rollback' approach. The 'run' method involves correcting the system's current state to maintain operation, while 'rollback' entails reverting the system to a prior stable state, such as through checkpoints, necessitating idempotent operations between checkpoints and the identified error. Some systems may employ both strategies, depending on the nature or segment of the error encountered.

Key features of a fault-tolerant system include:

- Multiple restoration points, allowing for failover control to be assumed by an alternate server if necessary;
  - Containment of damage to the affected component;
  - Prevention of fault propagation to other systems;
  - Availability of rollback options.

Fault tolerance is integral to business continuity, ensuring the high availability of computer systems and networks. Environments with fault tolerance can immediately restore services after a shutdown, whereas high-availability environments aim for near-constant availability, typically around 99.999% uptime.

High-availability clusters consist of independent server groups that collaborate to facilitate the exchange of vital data and resources across the system. These clusters

monitor one another's performance and intervene as needed to maintain application accessibility. On the other hand, a failover cluster involves multiple physical systems sharing a single server software instance, with commands executed on one system mirrored on the others.

Redundancy is the cornerstone of fault tolerance, achieved through strategies like data replication or synchronous volume mirroring to a secondary data center. Physical redundancy involves keeping additional hardware on standby for swift activation when required.

Data backup often goes hand-in-hand with redundancy, both serving to safeguard against data loss. While backups are generally focused on restorative processes over time, redundancy is tailored for applications with minimal tolerance for downtime. It's important to note that backups and redundancy serve different purposes and cannot substitute for one another. A robust failover system architecture should include regular backups of critical data, potentially complemented by mirroring to a backup or alternate server. Security measures must also be incorporated into the design to thwart unauthorized access.

5.3. Scalability and Fault Tolerance: Architecting Resilient Systems in the Digital Age

As one of the main conditions for building a fault-tolerant system is the presence of more than one restore point for each component, the development of such a component is relevant [13-14].

Scalability is an ability of a system to handle an increasing load, by system resources augmentation. In an economic context, a scalable business model assumes that a company can increase sales by increasing resources. In computer systems, scalability is a characteristic of computers, networks, algorithms, network protocols, programs and applications. For example, the search engine, which supports increasing the number of users and the number of topics it indexes.

Scalability can be measured in several measurements, such as [9]:

- Administrative scalability: the ability to increase the number of users to access the system;
- Functional scalability: the ability to improve the system by adding new functionalities without disrupting current activities;
- Geographic scalability: the ability to efficient support while area expands from local to larger;
- Load Scalability: the distributed system ability to expanding/contraction to handle larger/smaller loads, easely for component to be modified, added, or removed to meet variable loads;
- Generatoin scalability: the ability of a system to scale using new generations of components;
- Heterogeneous scalability: the ability to accept components from different vendors.

There are two types of scaling: horizontal and vertical [9]. Both types may be used, however, a properly designed system is scaled in two ways at once, it depends on each component task and needs.

Horizontal scaling means nodes adding/removing to/from the system, for example, adding a new computer or any other device to distributed software. Another example is scaling from one web server to three. High-performance computing programs (for example, the mathematical component of image processing [10, 11], code sequence processing [12], signal type conversion [13] and many other tasks that require high computing power) are scaled horizontally to support tasks that would require huge powers once. Very popular nowadays social networks, exceed the largest supercomputer powers and handled by scalable systems only. Using this type of scaling requires software to effectively manage and maintain resources.

Vertical scaling (up/down) means adding/removing resources to/from individual node (processors, RAM, or non-volatile memory). Vertical scaling is used when horizontal scaling requires more effort or money. As more elements, as management complexity increases, so some programs do not scale horizontally.

Database scalability requires that the database system be able to perform additional tasks considering larger hardware resources, such as additional servers, processors, and memory. Workloads continue to grow, and database requirements follow this trend. Algorithmic innovations include row-level locking and partitioning of tables and indexes. Architectural innovations include "nothing in common" and "all in common" architectures to manage multi-server configuration.

In context of data storage scaling, scalability is the maximum storage cluster size guarantees data consistency, ie there is only one valid version of stored data throughout the cluster, regardless of the number of redundant copies of data. Clusters that provide "lazy" redundancy by updating copies asynchronously are called "ultimately consistent." This type of scaling design is suitable when availability and responsiveness are rated higher than consistency, that is true for many file hosts or web caches (if you need the latest version, wait a few seconds for it to spread). This design should be avoided for all classic transaction-oriented applications.

The most of open and commercial storage clusters based on standard networks provide only possible consistency (NoSQL, CouchDB, etc.). Recording operations cancel other copies. Read operations usually do not check every redundant copy before answering, potentially there are no previous write operations. High metadata signal traffic requires specialized equipment and short distances to process with acceptable performance (ie acting as non-clustered storage devices or databases).

In high-performance computing field there are two general concepts of scalability. The first is strong scalability, which is defined as the change in decision time depending on the number of processors for a fixed total size of the problem. The second is weak scaling, which is defined as the change in decision time depending on the number of processors for a fixed task size per processor.

Autoscaling – is cloud computing method, so the amount of computing resources in the server farm (which is usually measured by the number of active servers) is automatically scaled based on the total farm load. Autoscaling is related to and relies on the idea of load balancing.

Autoscaling offers the following benefits: for companies that have their own web

server infrastructure, autoscaling usually means that some servers may "fall asleep" at low load, saving energy (as well as water if water is used as cooler). For companies with cloud-based infrastructure, autoscaling provides lower bills, as most cloud service providers charge based on general usage rather than maximum capacity. Even for companies that can't reduce the total computing power they perform or pay at any moment of time, autoscaling can helps by allowing companies to work on less time-consuming workloads on PCs (which are released by autoscaling during low loads). Autoscaling solutions (Amazon Web Services) can also provide replacement for problematic server instances, and thus provide some protection against hardware/software/network failures. Autoscaling can provide greater productivity and availability in cases where production loads are variable and unpredictable.

Autoscaling differs from having a fixed daily/weekly/annual server usage cycle because it responds to actual usage patterns, and thus reduces the potential disadvantage of having too few or too many servers to load traffic. For example, if traffic is typically lower at night, a static scaling solution may cause some servers to sleep at night, but this can lead to idle nights when people use the Internet more (for example, due to viral events, news). On the other hand, autoscaling allows you to better cope with unexpected load peaks.

Default autoscaling uses a reactive approach to decision-making to scale traffic: scaling occurs only real-time metrics change. In some cases, especially when changes occur very quickly, this reactive approach to scaling is insufficient, so the alternative is automatic scheduling or predictive autoscaling.

Scheduled autoscaling – changes are made due to the minimum/maximum size or desired autoscaling group power at an exact time of day. Scheduled scaling is useful if the known traffic load increases/decreases at an exact time of day, but the change is too sudden with reactive approach.

The autoscaling predictive approach uses predictive analytics. The idea is to combine recent usage trends with historical usage trends, as well as other types of data to predict future need for scaling.

5.4. Stateless Scaling: Optimizing Web Applications for Resilience and Efficiency

Web applications encapsulate core business logic and are ideally designed to be "stateless," meaning they do not retain user state within the application. To achieve this, certain optimizations are necessary for components that traditionally store state:

- Custom Files: User-uploaded files, if stored on the default file system, compromise the stateless nature of the application. To address this, files can be moved to a shared file system like GlusterFS or CephFS, which is straightforward to set up and doesn't require code changes. However, this can introduce delays in file distribution and potential issues under heavy loads. Alternatively, using a third-party service like Amazon S3 offers automatic file caching and efficient global delivery at a lower cost per gigabyte than traditional server storage, but it requires additional code to redirect file operations to the external service.
- Sessions: These are temporary data stores that maintain user identity across different parts of the application during their session. If a user's session is stored on Server_1 and they are switched to Server_2 by a load balancer, they may be inadvertently logged out. The solution is to store session data in a central database or a key-value store, with the latter often preferred for its speed and simplicity, as session data typically doesn't require the complex structure of relational databases.
- Shared System Files: Applications may generate specific files to signal various states, such as lock files. To maintain a stateless architecture, it's necessary to refactor the application to use external databases for state management instead of local files.

Once these considerations are addressed, the application is primed for containerization and scaling. The next step is to select an HTTP traffic balancing method:

- Load Balancer: Utilizing a load balancer like nginx can simplify server selection, filter out bad requests, and monitor the readiness of application

servers to handle incoming requests. The downside is the need for installation and configuration, and it typically requires a dedicated server.

– DNS Balancing: This simpler method assigns multiple servers to a single domain. While configuration options are limited, DNS balancing is useful for distributing traffic across multiple load balancers, especially when one is overwhelmed by high traffic.
– OSI Network Layer Traffic Balancing: This method operates at a lower level in the network stack, offering another approach to distributing traffic.

Each method has its advantages and trade-offs, and the choice will depend on the specific requirements and constraints of the application infrastructure (fig. 1).

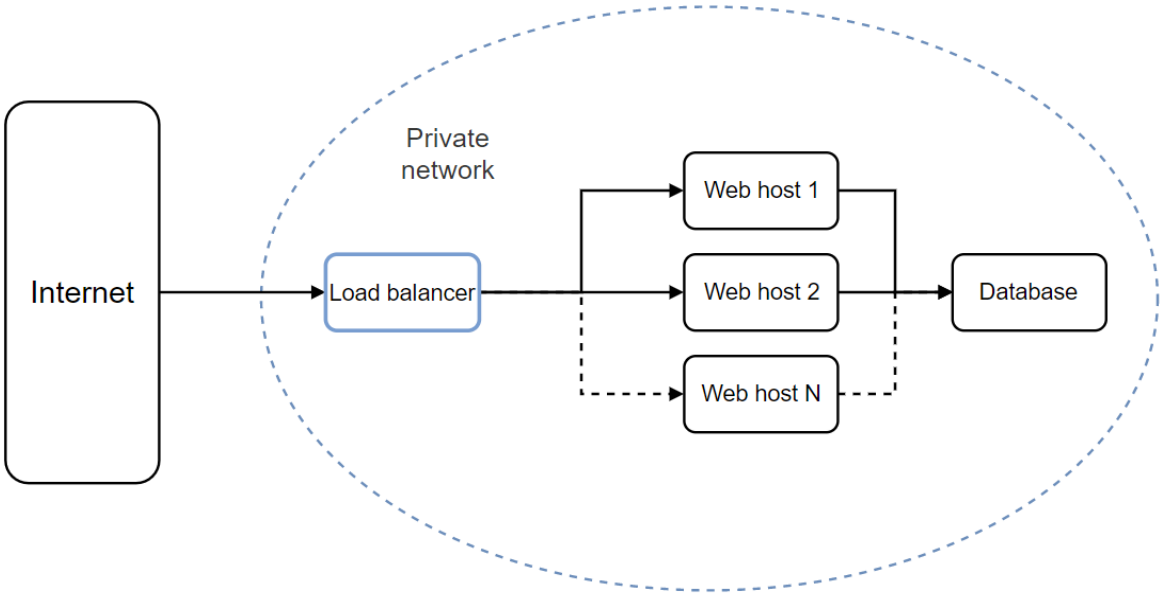

Figure 1 – The web-application uses Load balancer

Authoring

5.5. A combined databases scaling method

Databases inherently store state, including user and system data, necessitating support for data integrity and consistency during scaling—a complex task. The fundamental strategies for database scaling are replication and sharding.

Database Replication is the process of duplicating a database instance from a primary location to another. This technique enables the mirroring of a database from a

master DBMS to a slave database. There are two replication models:

- Master-Slave Replication: Chosen when the majority of database interactions are read operations. The master instance serves as the primary source, with its state replicated to secondary slave nodes. Writes are directed to the master, while reads can occur on both master and slave nodes. This model enhances database fault tolerance by maintaining multiple independent copies of the database. If a slave node fails, it is readily replaceable. Should the master fail, a slave node is promoted to master, and a new slave takes its place.
- Master-Master Replication: Allows both read and write operations on any instance. However, this model is less common and supported by fewer DBMSs.

These replication strategies are crucial for maintaining the continuous operation and reliability of database systems during scaling operations (fig. 2).

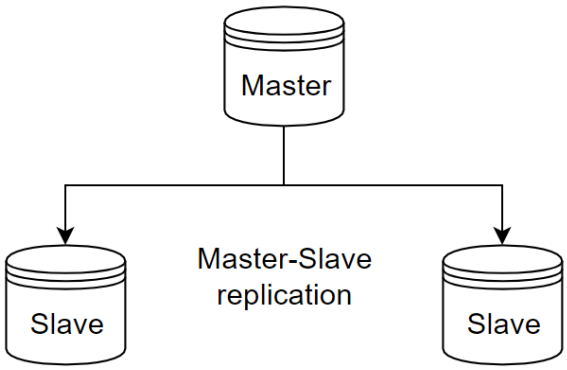

Figure 2 – DB instances interaction at Master-Slave replication

Authoring

Master-Master replication enables both reading from and writing to any database instance. However, this replication model is only supported by a select few database management systems.

Sharding is a technique where data is distributed across different database nodes. For instance, a database might be configured to hold information for up to 1000 users; once this threshold is reached, the application transitions to a new database instance.

These instances operate independently, and the application determines the appropriate node for each operation. Sharding operates autonomously, but to bolster the fault tolerance of individual shards, it's often paired with Master-Slave replication. This combination facilitates virtually limitless scaling of the database infrastructure.

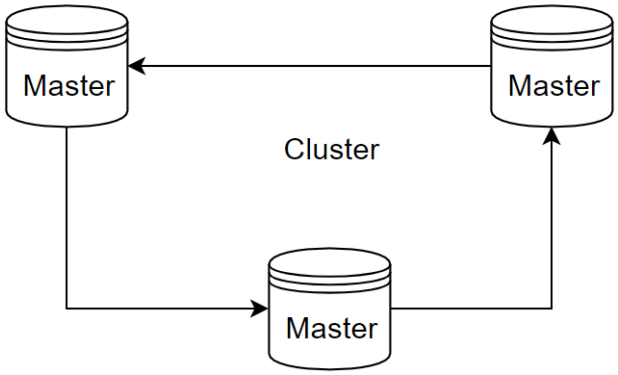

Figure 3 – DB instances interaction at Master-Master replication

Authoring

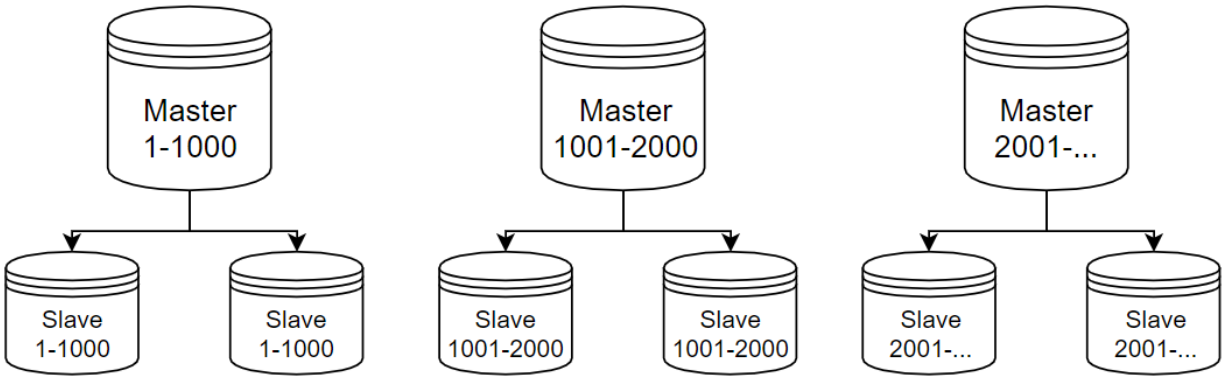

Figure 4 – DB instances interaction at combining Sharding and Master-Slave replication

Authoring

The scaling process for applications and databases varies based on the hosting environment, with autoscaling being a feature exclusive to cloud systems. The environments for deploying a web application include:

– Physical Server: Situated within the premises of the organization operating the web application, maintained by its own IT staff. Maintenance tasks encompass monitoring physical aspects like cooling and cleanliness, as

well as overseeing the operating system, software updates, and network security. Typically, a single operating system is installed, and applications vie for resources under heavy load conditions.

– Dedicated Server: Housed within a data center, the physical upkeep of a dedicated server is managed by the hosting provider. System configurations and application management are conducted remotely by the organization's administrators, allowing for flexibility in server location.

These environments dictate the scalability options and management practices for web applications, each with its own set of advantages and logistical considerations.

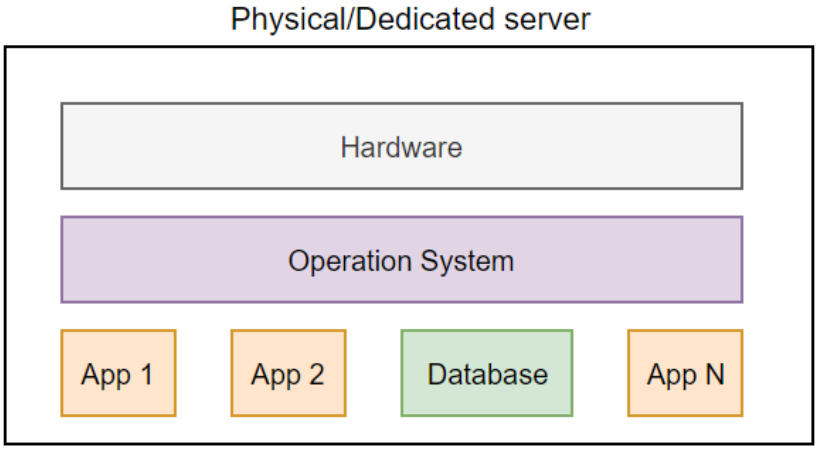

Figure 5 – Applications on a physical/dedicated server (applications compete for resources)

Authoring

Scaling applications on physical or dedicated servers typically involves either upgrading to a more powerful server or adding a new server and redistributing the applications. Both methods can lead to operational downtime and require careful planning.

In contrast, cloud environments, which can be public or private, offer a more flexible approach to application deployment. Public clouds provide services to anyone on a pay-per-use basis, while private clouds are bespoke solutions managed by a single enterprise. Cloud applications often run in containers, encapsulating each application in an isolated environment, which allows for independent scaling and adaptability to various capacities and settings (fig. 6).

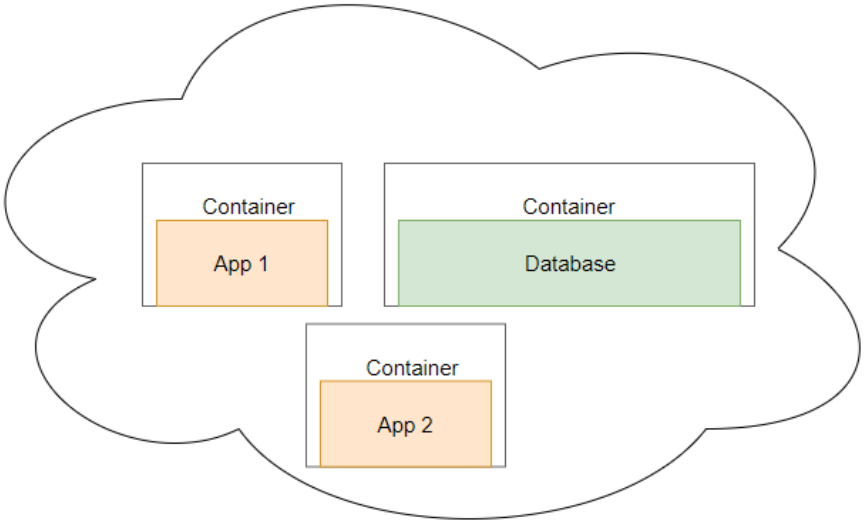

Figure 6 – Applications in the cloud (applications do not compete for resources)

Authoring

In cloud computing, the primary offering is virtual server leasing, complemented by services like database hosting, queue management, and document-oriented storage solutions. This setup allows for containerized application deployment on virtual servers, bypassing the need for individual database configuration and containerization. Cloud-based applications benefit from both horizontal and vertical scaling, which can be manually adjusted. Additionally, clouds feature autoscaling capabilities that can be tailored through the management console based on predefined criteria.

Service Mesh represents a higher level of abstraction, enabling the deployment of containerized applications and services without concern for their connection to the underlying physical infrastructure. It creates a virtual cluster from any number of servers, facilitating network, storage, and service discovery abstractions. It also oversees container lifecycles, ensuring their continuous operation and automatically restarting them if issues arise. Scaling within a Service Mesh is managed at the application integration level, with the system dynamically adjusting the number of active instances in response to fluctuating loads.

Within this ecosystem, applications and ancillary services collaborate to provide monitoring, logging, and command execution scheduling, all orchestrated through the Service Mesh framework (fig. 7).

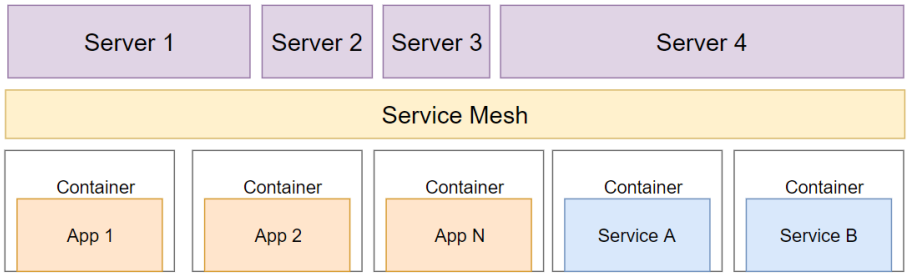

Figure 7 – Integration of applications and services in Service Mesh, a group of servers forms a single cluster, applications do not interact with servers directly

Authoring

Managing the size of a Service Mesh cluster is essential, especially when the current server group lacks sufficient capacity for scaling. This challenge is addressed by either expanding the number of active servers or adjusting the size of the server group, which typically remains on standby. When considering individual applications, this scaling is referred to as vertical; however, when applied to the cluster as a whole, it can be either vertical or horizontal, depending on whether the expansion involves adding more resources to existing servers or incorporating additional servers into the cluster.

5.6. Load balancing mechanism in the database

Upon thoroughly examining the deployment of web applications, methods for scaling services, and strategies for preparing projects for scalability, we can identify key comparative features. Since files must not reside within the application's file system to facilitate scaling, alternative file storage solutions must be selected. This work suggests a unified approach to database scaling that avoids the need for custom solutions for each project, thereby reducing development and maintenance costs (tab.1).

Table 1 – Saving custom files when running the application to provide scalability

Method	Integration	Record	Reading
GlusterFS / CephFS	Easy	Fast	Delivery delays between instances may occur
Amazon S3/Rackspace	Intensive	Recording delay may occur	Fast reading from anywhere in the world
Shared Volume Service Mesh	Easy	Fast	Relatively fast

Authoring

The comparative analysis indicates that utilizing a Service Mesh shared volume offers significant benefits, particularly as an optimal solution for managing shared user files.

When it comes to selecting a repository for custom sessions, Key-Value stores like Redis emerge as the preferred choice. Their scalability through clustering and high read/write speeds, attributed to in-memory data storage, make them highly effective.

The primary drawback is the requirement to implement and maintain a new service (tab.2).

Table 2 – Saving custom sessions

Method	Integration	Record	Reading	Scaling
Saving in a relational database (MySQL/PostgreSQL)	Easy	Slow	Slow	Time consuming
Saving to key-repository (Redis/Memcached)	Relatively easy	Fast	Fast	Easy

Authoring

The ways to host a web-application, you can see a comparison in table 3.

1) The cost associated with server units can escalate quickly for large-scale architectures that rely on physical or dedicated servers.
2) While some service providers offer these services, the scope for configuration is often restricted.
3) Service Mesh stands out for its exceptional flexibility in web application deployment, enabling rapid initiation and termination of clusters. This solution, while moderately priced, is cost-efficient due to its precise allocation of resources, ensuring no wastage.
4) Designed with high performance and reliability in mind, Service Mesh inherently incorporates autoscaling as a fundamental feature, facilitating efficient resource management.

Table 3 – Web-application placement methods

Method	Cost	Easy to set up	Flexibility	Autoscaling
Physical/dedicated server	Average	Difficult	Low	No
Virtual server	Low	Medium	Average	Conditionally absent
Cloud	Average	Simply	High	Yes
Service Mesh	Average	Medium	Highest	Works by default

Authoring

The analysis concludes that for objectives centered around high reliability, performance, and availability, Service Mesh technologies like Kubernetes are the superior choice. This approach requires a one-time configuration, after which it offers ongoing flexibility.

Additionally, it aligns with modern industry practices by containerizing applications and adopting the Infrastructure as Code (IoC) principle. When it comes to scaling a primary relational database, it's crucial to evaluate various aspects, including application load characteristics and data volume, to determine the most effective scaling strategy (tab.4).

Table 4 – Scaling the main database (MySQL/PostgreSQL)

Method	Integration	Ease of use	Fault tolerance	Installation
Master-Slave replication	Easy	Average	High	Relatively easy
Master-Master replication	Easy	High	High	Very difficult
Sharding	Medium	Average	Low	Medium
Sharding + Master-Slave	Medium	Average	High	Difficult

Authoring

This document suggests a refined load balancing approach across multiple database instances, enabling the deployment of numerous instances to guarantee both speed and fault tolerance under heavy loads. Upon reviewing various database scaling techniques for optimal stability and performance, the combination of "Sharding + Master-Slave Replication" emerges as the premier option. Despite its challenges in terms of integration and setup complexity, this method offers the ability to scale the database for an unlimited user base while preserving fault tolerance.

Summary and conclusions

This work explores mechanisms to enhance the resilience of standard web applications that utilize databases, and it addresses potential challenges that may arise while preparing to scale different components of an application. Scaling is the most effective strategy for constructing a fault-tolerant web application, and autoscaling can drastically cut down the time specialists spend on manual system configuration, as well as reduce the latency in responding to increased user traffic.

The database presents the most complex challenge in this process. Being stateful, it requires careful synchronization of distributed data to ensure consistency and availability. By employing the "Sharding + Master-Slave Replication" approach to scaling, it's possible to expand the database size indefinitely without compromising data access speed, while maintaining the capability for self-recovery in the event of issues.

For scaling web applications, Service Mesh is the optimal choice due to its ability to provide a sufficient margin of flexibility that can adapt to user needs over the long haul.

The work discusses the optimization and scaling of web applications that use databases, with a focus on scalability and fault tolerance:

- Deployment environments and autoscaling: The document compares different deployment environments, such as physical servers, dedicated servers, virtual servers, cloud systems, and Service Mesh, and highlights the advantages of autoscaling for improving system responsiveness and reducing manual configuration.
- Database management and scaling methods: The document examines the challenges of maintaining stateful databases and ensuring data consistency and identifies the "Sharding + Master-Slave Replication" method as an effective strategy for database scaling, allowing for potentially unlimited growth without sacrificing access speed or fault tolerance.
- Service Mesh as the best solution for web application scaling: The document recommends Service Mesh, particularly Kubernetes, as the optimal choice for scaling web applications due to its flexibility, cost-effectiveness, and alignment with modern practices like containerization and Infrastructure as Code (IoC). The document also suggests that Service Mesh provides the necessary infrastructure to support long-term user needs and is equipped with autoscaling as a core feature, making it the most advanced option for deploying and managing web applications.

The work concludes that scaling is the most effective strategy for constructing a

fault-tolerant web application, and that autoscaling can drastically cut down the time specialists spend on manual system configuration, as well as reduce the latency in responding to increased user traffic. The document also concludes that the combination of "Sharding + Master-Slave Replication" is the premier option for scaling the database, while Service Mesh is the superior choice for scaling the web application.