Issue 18, part 1, 2025

https://doi.org/10.23939/sisn2025.18.015

UDC 004.774

METHODS AND MEANS OF OPTIMIZATION OF DISTRIBUTED OLTP SYSTEMS

Oleh Faizulin[1]

[1] Lviv Polytechnic National University,
Department of Information Systems and Networks,
[1] oleh.r.faizulin@lpnu.ua, ORCID 0000-0001-5781-0600

The article explores the architecture of efficient and high-performance distributed online transaction processing systems, leveraging cloud-based tools, cloud-native architectural approaches, and database replication methods. It focuses on reducing network latency, optimizing resource usage−and consequently, costs−enhancing data replication, and improving fault tolerance. This article provides a practical demonstration of how modern cloud solutions and technologies enable the rapid and efficient development of enterprise-level distributed online transaction processing systems. The approaches discussed can be applied both to individual subsystems and as a comprehensive architectural strategy. The study examines key principles of system architecture design, selection of technologies to ensure performance and fault tolerance, and modern deployment methods for web applications. It highlights the importance of containerization and orchestration in simplifying infrastructure management. Additionally, it delves into automatic scaling mechanisms that dynamically adjust system resources in response to workload changes, ensuring optimal resource utilization. The proposed methods and approaches are relevant to developers, architects, and researchers working on building or optimizing modern OLTP applications. They provide valuable insights for building high-performance, scalable, and fault-tolerant systems.

Keywords: distributed web applications, online transaction processing, performance optimization, cloud computing, data replication.

Problem statement

Distributed web applications have become the backbone of modern computing, enabling scalable, high-availability services across geographically distributed environments. The increasing demand for real-time processing, data-intensive workloads, and global accessibility has pushed developers to design more efficient and optimized architectures. However, the complexity of distributed systems introduces various challenges, including latency issues, resource management, scalability constraints, and fault tolerance.

To maintain key non-functional requirements, cloud-native architecture must be utilized. At the same time, optimization techniques must be implemented at multiple levels, including, but not limiting to, infrastructure, application, and data layers. These techniques include DNS routing, load balancing, caching mechanisms, and efficient database management. Each of these plays a critical role in enhancing the overall efficiency of online transaction processing (OLTP) web applications.

Furthermore, as cloud computing and edge computing evolve, new optimization opportunities emerge. Serverless computing, container orchestration, and performance monitoring offer new ways to improve response times and resource utilization dynamically. This article provides a comprehensive analysis of optimization methods for modern OLTP web applications, exploring both traditional and emerging strategies.

As OLTP web applications grow in complexity, several challenges arise that can impact performance, reliability, and user experience. The primary issues include latency, where the geographical distribution of application components can increase response times and lead to performance degradation, with network latency, API call overhead, and inefficient data transfer mechanisms contributing to this problem. Another challenge is resource utilization, since many applications suffer from inefficient usage of computational resources such as CPU, memory, and bandwidth, and without proper optimization, resource over-provisioning or underutilization may result in redundant operational costs. Scalability limitations also play a significant role, as handling an increasing number of concurrent users requires robust scaling strategies. Finally, fault tolerance is crucial for distributed applications, which must be resilient to hardware failures, network disruptions, and software crashes, and ensuring high availability and minimizing downtime requires effective redundancy, replication, and failover mechanisms.

To address these challenges, a combination of architectural techniques is typically used. They include GeoDNS routing, load balancing, scalable deployments, database optimizations and many others. The article proposes a reference architecture for the geographically distributed OLTP systems that holistically address all above-mentioned issues. At the same time, each proposed technique can be used in a standalone manner, to address specific problems of the existing system.

Analysis of Recent Studies and Publications

Exploring the specific challenges of OLTP systems in cloud environments, Haubenschild et al. (2021) in their article "OLTP in the Cloud: Architectures, Tradeoffs, and Cost" provide a significant contribution by dissecting various architectural approaches for deploying OLTP workloads within cloud environments. The authors analyze both "lift-and-shift" migrations of traditional systems and the adoption of cloud-native database services, such as Amazon Aurora which is discussed in this article. Their work meticulously evaluates the tradeoffs involved concerning performance, scalability, availability, durability, and particularly operational costs. This is highly relevant as it underscores the importance of architectural choices for attaining cost-effective and high-performance OLTP systems, a key goal of the methods proposed herein. The insights from Böhm et al. reinforce the need for a thorough evaluation of database service models and their implications for overall system efficiency and resource utilization.

The article titled "Deploying and Managing Web Application Using Kubernetes" by Dhanwai, Bhagwat, Supekar, Deshmukh, and More, presented at the 2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), targets methodologies and best practices for deploying and operating web applications using Kubernetes. As one of the most wide-spread container management solutions, Kubernetes has become integral for automating the rolling-out, operating and scaling of applications. The authors delve into the core components of Kubernetes, such as Pods, Deployments, Services, and ConfigMaps, illustrating how these elements collaborate to ensure seamless deployment and management of web applications. They provide practical insights into creating Deployment manifests to manage application instances and utilizing Services to expose these applications to external traffic.

The article by Galipelli, S., Banka, M., & Banka, R. (2025) titled "Building A Serverless Application Using AWS Lambda" explores the development of serverless applications leveraging AWS Lambda, a key service provided by Amazon Web Services (AWS) that allows developers to run code without requiring of dedicated servers. This article serves as a practical guide for developers and architects looking to adopt serverless computing with AWS Lambda, offering both theoretical insights and hands-on implementation strategies.

The paper titled "Architecting Multi-Cloud Applications for High Availability using DevOps" by Damien Gallagher and Ruth G. Lennon (2022), presented at the 2022 IEEE International Conference on E-Business Engineering (ICEBE), addresses the design and implementation of multi-cloud applications with a focus on high availability through DevOps practices. This work provides insights into leveraging multi-cloud architectures and DevOps methodologies to build robust applications capable of maintaining high availability in dynamic cloud environments.

In his paper, "Amazon Aurora: Insights and Benchmarks for Contemporary Application Scaling," Goel R. (2025) examines Amazon Aurora's architecture and performance to highlight its effectiveness in modern application scaling. Functioning as a high-performance and fully managed relational database solution, Aurora combines the simplicity of open-source database systems with the performance of commercial equivalents.

In his work "Cloud Native Architecture and Design Patterns," Shivakumar R. Goniwada (2021) delves into the principles and methodologies essential for building cloud-native applications. This section, spanning pages 127 to 187, offers a comprehensive exploration of design patterns that facilitate the development of scalable, resilient, and efficient cloud-native systems.

The article by Khanal, D. D., & Maharjan, S. (2024) titled "Comparative Security and Compliance Analysis of Serverless Computing Platforms: AWS Lambda, Azure Functions, and Google Cloud Functions" examines the security and compliance measures used by most popular serverless computing platforms. This study is valuable for cloud architects, DevOps teams, and security professionals selecting a serverless platform based on security posture and regulatory requirements. It goes beyond feature lists to offer a risk-based comparison.

The article by Lima et al. (2023) titled "Efficient Causal Access in Geo-Replicated Storage Systems", published in the Journal of Grid Computing, investigates optimizing causal consistency models in geographically distributed (geo-replicated) storage systems. This work advances the design of high-performance geo-replicated systems, offering a pragmatic solution for scenarios where strong consistency is overkill, but eventual consistency is too weak.

The book “Kubernetes in Action”, Lukša M. (2022) provides a comprehensive guide to developing and running applications within a Kubernetes environment. It reviews container technologies, such as Docker, ensuring readers understand how to build and manage containers. Lukša then delves into Kubernetes, teaching readers how to deploy container-based distributed applications effectively. The book covers setting up Kubernetes clusters, managing applications, and understanding Kubernetes' architecture and operation. It also addresses advanced topics like monitoring, tuning, and scaling applications within Kubernetes.

In their paper, "Cloud Computing Based Learning Web Application Through Amazon Web Services," Neela et al. (2021) explore the development of an advanced E-Learning Management System (E-LMS) leveraging Amazon Web Services (AWS). The study highlights the integration of AWS's cloud infrastructure to create a scalable, secure, and efficient online educational platform. This research underscores the potential of cloud computing, particularly AWS, in transforming traditional e-learning platforms into dynamic, flexible, and secure environments that cater to the evolving needs of modern education.

These and other papers provide an invaluable source of knowledge and information regarding the problem.

Formulation of the Article’s Objective

The article aims to provide the architecture approach for a highly distributed enterprise-grade OLTP system. To achieve this, the article utilizes a set of technologies provided by AWS (Amazon Web Services) as reference technology stack. While AWS is a widely adopted and feature-rich platform, it’s not necessary to utilize AWS technologies at all. Each layer can be replaced with an alternative of GCP (Google Cloud Platform, MS Azure) or other vendors. The article targets to resolve the following issues out-of-the-box by utilizing the proposed architecture:

- Latency issues
- Resource utilization and application runtime scaling
- Database scalability and consistency
- Fault tolerance

By addressing these key challenges, this article provides a comprehensive guide to building cloud-native, optimized distributed web applications, ensuring enhanced performance, reliability, and security in modern environments.

Main Results

Latency Issues

Latency stands as a critical performance indicator for distributed web applications because it markedly influences both user satisfaction and overall system productivity. This term describes the time lag experienced between a user's action and the corresponding system reaction, a duration susceptible to multiple influencing elements. Latency can stem from various origins, such as ineffectively written application code, poor database query execution, a shortage of necessary resources, or delays in server-side processing. Although numerous such problems are solvable via code enhancements and resource governance across different layers of an application design, network-induced latency presents an ongoing difficulty. Unfortunately, unlike other performance hindrances, network latency cannot be entirely nullified solely through server-side or application-level tuning efforts; however, its effects can be lessened by strategies that work in conjunction with Geographic Domain Name System (GeoDNS).

To address network latency challenges, edge computing has surfaced as a potent methodology. By situating data processing nearer to the end-user−at the network's "edge"−the physical span data must traverse can be considerably shortened. This closeness to users facilitates quicker system reactions and heightened performance, particularly for applications demanding immediate, real-time engagement. Essentially, an optimal performance is achieved when requests are handled by the closest feasible proximity to the end-user, thereby minimizing delays associated with network data transmission.

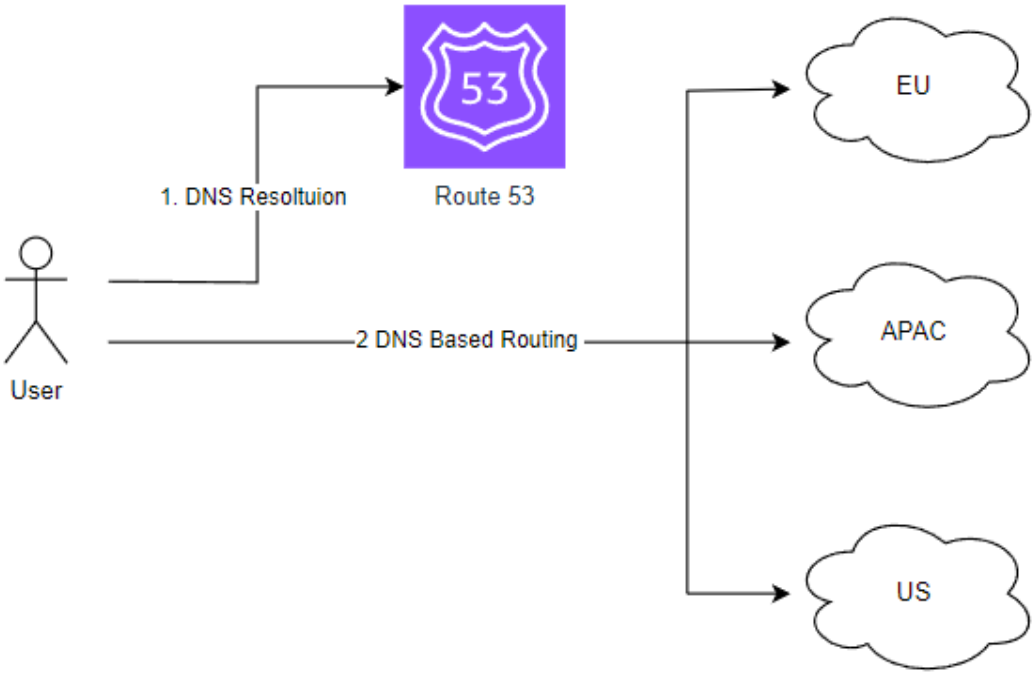

Fig 1. Location aware DNS-based routing

A primary strategy involves employing what is known as GeoDNS. Ordinarily, to decrease latency within a particular geographic zone for enterprise applications, an application regional duplicate established. Subsequently, the GeoDNS configuration is modified by adding records pertinent to that specific region. Illustrating this, Route53 serves as one such system for managing GeoDNS functionalities. Other available competing services include GoDaddy Premium DNS, Azure DNS, GCP DNS, and Cloudflare DNS, among others.

	Barcelona	Paris	Tokyo	Toronto	Washington
Amsterdam	28.831ms	10.533ms	227.388ms	87.638ms	79.794ms
Auckland	302.256ms	259.816ms	206.736ms	199.047ms	187.591ms
Copenhagen	46.137ms	21.938ms	259.943ms	100.164ms	108.902ms
Dallas	140.172ms	109.964ms	135.831ms	34.44ms	59.169ms
Frankfurt	23.451ms	13.095ms	251.645ms	97.997ms	86.584ms
London	27.103ms	7.545ms	230.431ms	82.402ms	74.054ms
Los Angeles	168.517ms	142.029ms	107.59ms	67.961ms	72.771ms
Moscow	85.289ms	54.544ms	273.394ms	130.895ms	117.751ms
New York	90.408ms	71.948ms	171.948ms	16.058ms	6.439ms
Paris	18.982ms	—	246.427ms	91.154ms	77.912ms
Stockholm	53.991ms	31.509ms	257.331ms	118.333ms	97.719ms
Tokyo	274.517ms	246.557ms	—	171.068ms	148.956ms

Fig 2. https://wondernetwork.com/ ping statistics

Finally, it must be mentioned network latency can be caused by other factors: low throughput, DDoS attack, insufficient resource, etc. It's out of scope for the article to address such issues.

Resource Utilization

Efficient resource utilization is a must for optimizing the performance, cost, and scalability capabilities of OLTP applications. In cloud-based and on-premises distributed environments, ineffective allocation and management of computational resources leads to increased latency, unnecessary costs, and system inefficiencies. By implementing intelligent resource management strategies, organizations can ensure that applications run in an optimized manner while minimizing overhead and maximizing throughput. Modern OLTP applications are typically deployed in using the following approaches:

- Virtual Machine (e.g. EC2)
- Containerized deployment (e.g. Kubernetes)
- Serverless deployment (e.g. AWS Lambda)

Barebone physical hardware deployments are rare and may be omitted due to being irrelevant for most enterprises. The table below illustrates benefits and drawbacks of each major deployment type.

Comparison of the Hosting Approaches

	EC2	Kubernetes	Lambda
Workload Type	Virtual Machine	Container Runtime	Serverless
Resource Overhead	High	Medium	Low
Cost	Fixed	Low	Medium
Scalability	Low	High	Extreme

To make an informed decision, it is crucial to analyze the application's behavior and workload patterns. However, when considering a typical OLTP application, a containerized runtime - specifically Kubernetes - is one of the best possible choices. This preference is driven by several key factors:

- Enterprise Workloads: Large-scale enterprises rarely operate a single application in isolation. Instead, they deploy a set of applications that work in an interconnected manner. By leveraging a container orchestration platform like Kubernetes, organizations can establish a unified cross-cluster configuration, optimize resource allocation, and enable rapid scaling, ensuring efficient workload management.

- OLTP-Specific Characteristics: OLTP applications are characterized by handling a high volume of concurrent, short-lived transactions. These applications require a runtime environment capable of efficiently processing multiple requests in parallel while maintaining low latency. Kubernetes, with its feature of managing containerized workloads dynamically, is well-suited to support such demanding workloads.

From a cost and performance perspective, Kubernetes generally proves to be a cheaper and scalable solution compared to alternative runtime environments. In scenarios with high concurrency, Kubernetes often outperforms serverless solutions like AWS Lambda in terms of cost efficiency while offering superior scalability compared to traditional virtual machines (such as EC2 instances). Furthermore, Kubernetes provides a robust set of built-in functionalities essential for both engineers and system operators. These include automated load balancing, security policy enforcement, advanced scalability mechanisms, monitoring capabilities, and seamless integration with cloud-native tools. By choosing Kubernetes for OLTP applications, organizations benefit from a highly adaptable, cost-effective, and operationally efficient runtime environment.

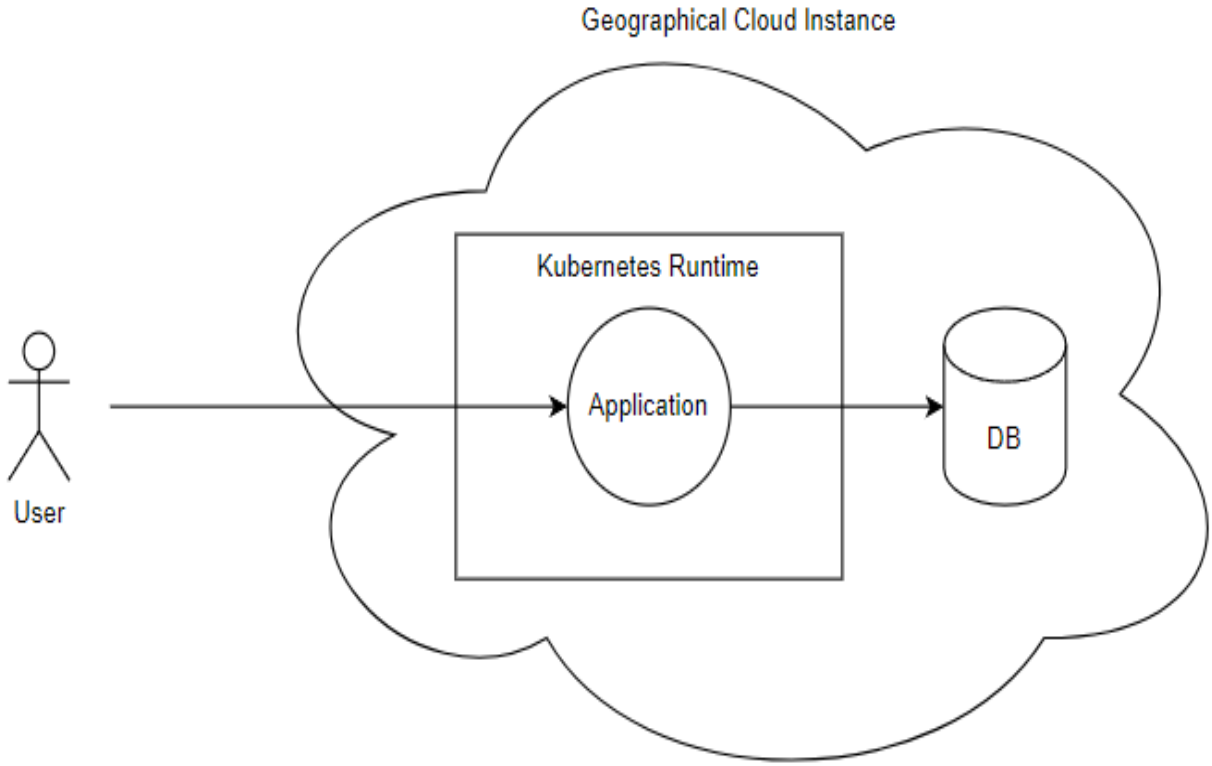

Fig 3. A typical OLTP system regional cluster setup

Effective Scalability of Application Runtimes

Scalability is one of the most important factors in the architecture of distributed web applications, enabling systems to efficiently manage increasing workloads without performance degradation. As user demands and data volumes grow, applications must dynamically allocate resources to remain responsive and reliable. However, many distributed systems encounter challenges that impede seamless scaling, such as inefficient resource reservation, bottlenecks in processing, and excessive delays in response times. These issues can result in reduced throughput, higher latency, and suboptimal utilization of available computing power. To mitigate these challenges, it is essential to apply a well-structured application configuration that balances resource allocation effectively. A misconfigured application may suffer from resource stagnation, where it lacks the necessary computational power to perform optimally. Conversely,

improper allocation leads to resource wastage, whereas the system allocates more computing capacity than it can efficiently utilize. A carefully designed configuration ensures that the application scales efficiently without unnecessary performance trade-offs.

To illustrate the impact of scaling strategies, let's compare two Kubernetes deployment configurations.

```yaml
spec:
  replicas: 3
  template:
    spec:
      containers:
          resources:
            requests:
              cpu: "1"
            limits:
              cpu: "1"
```

```yaml
spec:
  replicas: 1
  template:
    spec:
      containers:
          resources:
            requests:
              cpu: "3"
            limits:
              cpu: "3"
```

Fig. 4. 3x1 CPU core vs 1x3 CPU core K8S configuration

The first configuration deploys three replicas of the application, with each replica utilizing one full CPU core. The second configuration, in contrast, deploys a single instance of the application but allocates three CPU cores to that instance. By examining these two approaches, we can evaluate how resource distribution affects throughput, resource utilization, scalability, and efficiency in handling concurrent workloads.

To further explore the implications of these configurations, let's evaluate their performance using following runtimes: Java, Node.js, and Python. Java applications are inherently multithreaded, allowing them to efficiently utilize multiple CPU cores by executing concurrent tasks within a single process. This makes the second configuration - where a single instance has multiple cores - highly suitable for Java-based applications, as it enables efficient parallel execution and minimizes the overhead of inter-instance communication. On the other hand, Node.js and Python primarily operate in a single-threaded environment, relying on asynchronous execution rather than true parallelism. Since these runtimes do not natively distribute tasks across multiple CPU cores, assigning multiple cores to a single instance does not yield substantial performance improvements. Instead, this results in inefficient resource utilization, as only one core may be actively used while others remain idle. Therefore, for Python and Node.js applications, distributing the workload across multiple instances - as seen in the first configuration - provides better scalability and resource efficiency.

In summary, selecting an appropriate scaling strategy requires a deep understanding of an application's runtime characteristics. The first configuration, which distributes workloads across multiple instances, works best with single-threaded applications like Python and Node.js, where horizontal scaling (replicating instances) provides better performance. Meanwhile, the second configuration, which consolidates resources into a single instance, is better aligned with multithreaded applications like Java, which can effectively leverage multiple CPU cores. Ensuring that the chosen scaling model aligns with the runtime's capabilities is crucial for optimizing performance, maximizing resource efficiency, and maintaining a cost-effective deployment strategy in a Kubernetes-based environment.

Database Scalability and Consistency

Data consistency is yet another important characteristic of distributed applications. It ensures that users receive accurate, up-to-date information, regardless of their geographical location and selected cluster instance. In geographically distributed applications, maintaining strong consistency can be

challenging due to network latency, partitioning, and the requirement of synchronization across multiple regional cloud instances. To resolve these issues, many large-scale systems adopt an eventual consistency model, which prioritizes availability and performance while allowing temporary inconsistencies that resolve over time. While this approach improves scalability and fault tolerance, it requires careful design to handle scenarios where data may be briefly out of sync, ensuring a seamless and reliable user experience. There are multiple solutions and approaches to database scalability and consistency which are a subject for a separate research project. For illustrative purposes, let's compare two representatives of database systems: DynamoDB for NoSQL and Aurora DB as a relational database.

DynamoDB utilizes eventually consistent reads by default and prioritizes availability and low latency. When the data is written, it's asynchronously replicated across multiple regions, meaning a read request may return stale data for a short time frame. Optionally, DynamoDB allows strong consistency, what make sures that the reading operation always returns the last committed write. This requires querying the leader node directly, which may introduce a higher latency, especially in multi-region deployments.

Amazon Aurora utilizes strong consistency for the region and eventual consistency for global distribution. Whenever working within region, every write operation is synchronized across at least six copies of the data in three availability zones, ensuring strong consistency across replicas. Once transaction is confirmed, the data is immediately available for reading. At the same time, cross-regions synchronization is eventually consistent.

To sum it up, achieving cross-region consistency involves balancing consistency, availability, and partition tolerance Consistency, Availability, Partition tolerance theorem (CAP theorem). While strong consistency provides certainty, it can reduce system performance, while eventual consistency improves scalability and fault tolerance but introduces the risk of temporary data inconsistencies.

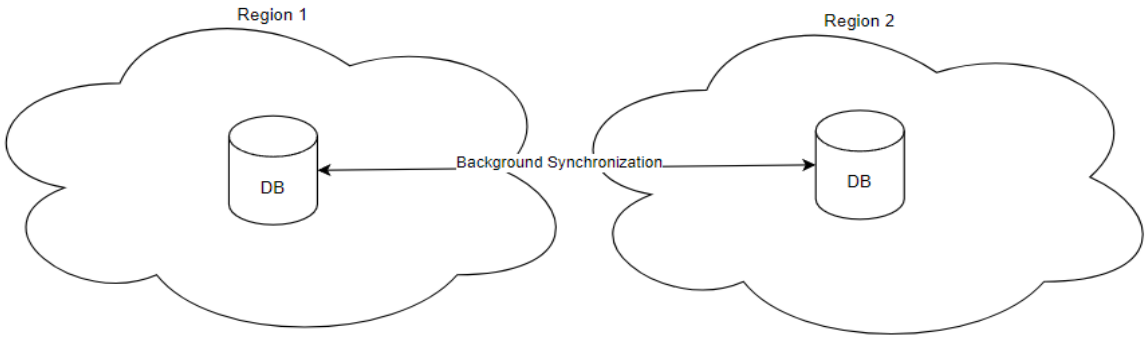

Fig 5. Background synchronization between databases

Fault Tolerance

In today's ever-evolving digital landscape, achieving fault tolerance has become significantly easier and more efficient using modern cloud systems like AWS. These cloud technologies are designed with built-in redundancy, automatic failover mechanisms, and highly distributed infrastructures, which allows relatively simple creation of fault-tolerant and resilient systems that can handle hardware failures, network issues, and other types of disruptions with minimal impact. The advanced capabilities provided by cloud platforms make fault tolerance an inherent feature, simplifying the overall design, setting a solid availability baseline and ensuring that applications continue to function smoothly even when failures occur.

Conversely, constructing a fault tolerant system without utilizing cloud technologies presents a far more complex challenge. Without the robust infrastructure provided by the cloud, enterprises have to manually design and manage their own redundancy and failover systems across multiple data centers or physical locations, which is not only resource-intensive but also prone to human error. Traditional on-premise solutions often require a significant investment in hardware, expertise, and operational maintenance, which can make building and maintaining fault-tolerant systems a daunting task for many businesses.

To ensure high availability within cloud environments, selecting appropriate services and components is essential that meet the specific needs of the application. Selecting the appropriate compute, storage, and networking services is critical to achieving the desired level of fault tolerance. For instance, offerings such as Amazon EKS, Amazon S3, and Amazon RDS come with built-in redundancy and failover capabilities which can be utilized for high availability. However, selecting the right configuration specific to each service, which involves replication strategies, backup policies, and monitoring tools, is key to ensuring resilience. Calculating the total uptime based on these selected services and understanding the associated Service Level Agreements (SLAs) can offer a distinct understanding of the system's overall reliability and help in identifying potential vulnerabilities.

Additionally, it is likewise crucial to utilize cloud-native architectural patterns while building an application. Cloud-native applications are built to fully capitalize on the scalability, elasticity, and fault tolerance features offered by cloud platforms. This involves utilizing containerization (such as Docker) and orchestration tools (like Kubernetes) and ensuring that the application is capable of dynamically scaling to meet demand and upholding fault tolerance. Employing cloud-native patterns and leveraging the full potential inherent in cloud platforms, organizations can make sure that their applications extend beyond fault tolerance to also be highly scalable, efficient, and resilient in the face of potential failures. Through this approach, businesses can maximize uptime and ensure optimal performance, even amid unforeseen disruptions.

Conclusions

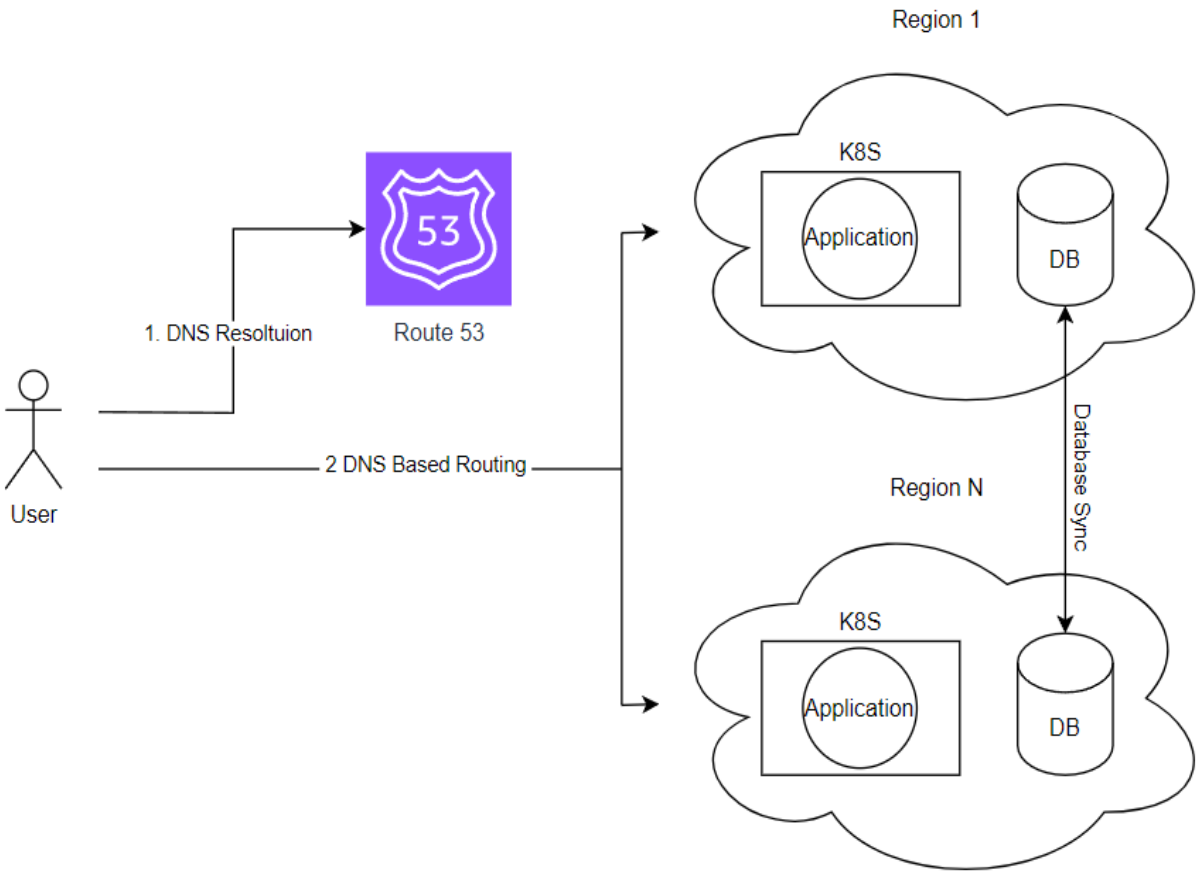

Fig 6. Suggested Reference Application Architecture

In the modern era, distributed OLTP applications are essential for any enterprise or large-scale business. At the same time, it's both challenging and easier than ever to build a scalable, distributed, cloud-native OLTP system. The key takeaways are:

- Utilize GeoDNS like Route53 for automatic DNS resolution, cross-region routing and failover

- Utilize containerized application packaging, for instance docker, and container runtime, like Amazon EKS
- Utilize cross-region data replication by selecting proper database solution like DynamoDB or Aurora DB.
- The fundamental cloud-native architectural principles discussed apply whether the application is a monolith or composed of microservices, though the implementation details and complexity may vary.
- Finally, don't forget about system monitoring and alerts

By following these simple principles and reference architecture any modern OLTP application can be deployed with high availability, fault tolerance and low latency.

REFERENCES

Dhanwai, A., Bhagwat, S., Supekar, K., Deshmukh, S., & More, A. (2025). Deploying and Managing Web Application Using Kubernetes. 2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), 130–136. https://doi.org/10.1109/idciot64235.2025.10914907

Dibya Darshan Khanal, & Sushil Maharjan. (2024). Comparative Security and Compliance Analysis of Serverless Computing Platforms: AWS Lambda, Azure Functions, and Google Cloud Functions. Research Square (Research Square). https://doi.org/10.21203/rs.3.rs-4823011/v1

Galipelli, S., Banka, M., & Banka, R. (2025). Building A Serverless Application Using AWS Lambda. https://doi.org/10.2139/ssrn.5068707

Gallagher, D., & Lennon, R. G. (2022, October 1). Architecting Multi-Cloud Applications for High Availability using DevOps. https://doi.org/10.1109/ICEBE55470.2022.00028

Goel, R. (2025). Amazon Aurora: Insights and Benchmarks for Contemporary Application Scaling. International Journal of Computer Applications, 186(70), 29–31. https://doi.org/10.5120/ijca2025924554

Goniwada, S. R. (2021). Cloud Native Architecture and Design Patterns. Cloud Native Architecture and Design, 127–187. https://doi.org/10.1007/978-1-4842-7226-8_4

Haubenschild, M., & Leis, V. (2025). Oltp in the cloud: architectures, tradeoffs, and cost. The VLDB Journal, 34(4). https://doi.org/10.1007/s00778-025-00913-z

Lima, S., Filipe Araújo, Miguel, Correia, J., Bento, A., & Barbosa, R. (2023). Efficient Causal Access in Geo-Replicated Storage Systems. Journal of Grid Computing, 21(1). https://doi.org/10.1007/s10723-022-09640-z

Marko Lukša. (2022). Kubernetes in Action, Second Edition. Manning.

Neela, S., Neyyala, Y., Pendem, V., Peryala, K., & Kumar, V. V. (2021, March 1). Cloud Computing Based Learning Web Application Through Amazon Web Services. https://doi.org/10.1109/ICACCS51430.2021.9441974

МЕТОДИ ТА ЗАСОБИ ОПТИМІЗАЦІЇ РОЗПОДІЛЕНИХ СИСТЕМ ОНЛАЙН-ОПРАЦЮВАННЯ ТРАНЗАКЦІЙ

Олег Файзулін[1]

[1] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна,
[1]E-mail: oleh.r.faizulin@lpnu.ua, ORCID: 0000-0001-5781-0600

Стаття досліджує архітектурні рішення для ефективної та швидкої побудови розподілених систем онлайн опрацювання транзакцій із використанням хмарного інструментарію, cloud-native архітектурних принципів, та методів реплікації баз даних. Стаття фокусується на методах

зменшення мережевої затримки, оптимізації використання ресурсів і, як наслідок, коштів, реплікації даних та відмовостійкості. Стаття наглядно демонструє як із використанням сучасних хмарних рішень та технологій можна швидко та легко побудувати розподілену систему онлайн опрацювання транзакцій корпоративного рівня. Рішення використані у статті можуть бути застосовані як для окремих підсистем, так і як цілісний архітектурний підхід. Розглядаються принципи побудови архітектури систем, вибір технологій для забезпечення продуктивності та відмовостійкості. Також аналізуються сучасні методи розгортання веб-додатків, включаючи використання контейнеризації та оркестрування для спрощення управління інфраструктурою. Окремо розглядаються механізми автоматичного масштабування, що дозволяють динамічно адаптувати систему до змін навантаження, оптимізуючи використання ресурсів. Запропоновані методи та підходи є актуальними для розробників, архітекторів та дослідників, які працюють над оптимізацією розподілених веб-додатків та прагнуть створювати високопродуктивні, масштабовані й стійкі до відмов системи.

Ключові слова: Розподілені веб-додатки, система онлайн опрацювання транзакцій, оптимізація продуктивності, хмарні обчислення, реплікація даних.