International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Isolation Techniques for Preventing Cascading Failures in Multi-Tenant Multi-Cluster Environments

Anila Gogineni

Independent Researcher, USA

ABSTRACT

This paper discusses measures that can be employed to avoid domino effects in multi-tenant, multi-cluster architecture that is common in contemporary cloud systems. This paper aims at reviewing logical, physical, resource, fault, and network isolation techniques with a view of ascertaining how they curb failure domino effects across tenants. These advanced techniques demonstrate the necessity and care to be taken while choosing and deploying these techniques so as to combine low error handling with system performance. Finally, some suggestions about distance semantics implementation for large-scale multi-tenant clouds to increase overall system irresponsibility and reduce the adverse effects of failures in large-scale Cloud environments are provided.

Keywords: Isolation Techniques, Cascading Failures, Multi-Tenant, Multi-Cluster, Fault Isolation, Distributed Systems, Cloud Computing, Resource Contention, Network Isolation, Virtualization

Introduction

Multi-tenant and multiple-cluster are two important parts of most of the current cloud architectures that allow resource sharing between users and applications. Multi-tenancy is a physical architecture where one physical installation of the application can support multiple customers (tenants) with the same data and computing resources but are logically separated from each other. Like Multi-Cluster architecture which employs several clusters that are designed solely for tackling workload distribution, resources management, and failures. These configurations are commonly used because they are scalable, inexpensive or cheap as well as elastic in cloud platforms.

Yet, multi-tenant and multi-cluster arrangements present risks such as multiple tenants or clusters suffering from the cascading failures at sites and services. Such failures can then spread throughout the system via interconnected resources, thus touching on issues of performance, availability and vulnerability. For instance, a problem in one tenant's application may be due to resource competition and which results in poor performance in other tenants that are using the same resources. This can lead to user inconvenience, reduction in the availability of service and increased cost of system operation.

Several distinct approaches to isolation will be investigated in this research. These approaches have the potential to assist in preventing chain reactions in these adaptable systems. The capacity to isolate must be of utmost importance in order to prevent issues that arise with specific tenants or clusters of renters from spreading to other tenants. It is feasible for system administrators to maintain their networks as safe

as possible with the assistance of effective isolation methods, which also help them decrease the amount of time they spend correcting defects and efficient resource management.

The report covers a wide range of topics, some of which include the following: a description of the dangers associated with cascading failures; an overview of the methods of isolation, including logical isolation, physical isolation, and isolation at the network level; and an evaluation of the effectiveness of the methods mentioned in relation to the potential dangers that have been elaborated. In addition, the report will include some instances taken from real life as well as evaluations from experts about the effectiveness of certain methods.

Background and Related Work

Multi-tenant and multi clusters are the key features of cloud computing; it offers a flexible approach for organizing the resources and applications for numerous users. In a multi-tenant environment different customers (tenants) use the same underlying physical resources to compute, store and communicate, but partitioned in a way to ensure that each tenant's data and applications are separate from those of others. Pleased with this concept, multi-cluster systems go further by placing workloads across multiple clusters in different geographic data centers or computing nodes, thereby improving scalability and dependability. These architectures are to accommodate various workloads, traffic intensity and different users' needs [1]. However, such architectures DG Preferred and RI5G are unified which makes them vulnerable to issues such as cascading failure. Cascading failures are a condition whereby, when one tenant or cluster develops a fault, it has a knock effect on the other part of the system. For example, an authorized resource churn or a security violation in one tenant can trigger resource contends in others that will result in other tenants or clusters being slowed down or halted altogether. This chain reaction results in potential system unavailability, cost overruns and loss of reputation; which is why the mitigation of cascading failures is important for system integrity.

Previous works have also recommended several approaches for isolation in order to prevent cascading failures [2]. Virtualization and containerization of data guarantee that the workloads are secured to work within the same infrastructure without interference that can affect the other tenets. Physical separation ensures that each tenant has its own hardware, failure containment is also limited to one tenant only. Whereas quotas and rate-limiting controls confine resource usage, network isolation using VPNs or SDN guarantees secure data sharing. Moreover, component fail containment strategies such as redundancy and replication reduce the effects of failures. Despite these successes, difficulties in maintaining scalability, fault-tolerant capacity, and providing the optimal resource usage still remain big at large-scale levels. Scholars are developing dynamic isolation and machine learning-based methods to detect and prevent failures on the fly.

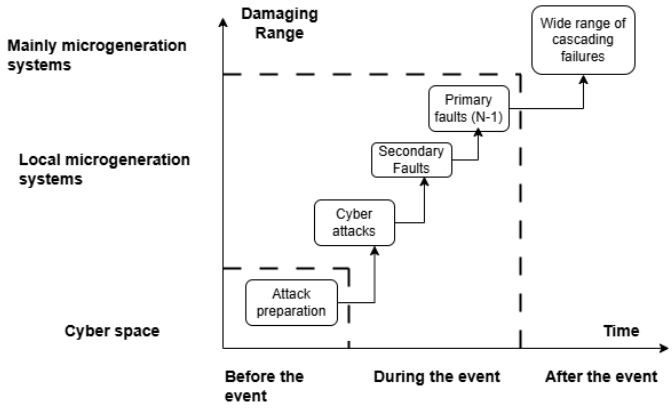

Fig 1: Cascading Failure

The next diagram presents a multi-cluster environment and demonstrates how failure can affect tenants and propagate between them if proper isolation mechanisms are not implemented.

Problem Statement and Challenges

Multiple failures in the multi-tenant and multi-cluster environments are some of the biggest challenges of cloud infrastructure. These systems are interconnected in such a way that when one tenant or cluster goes down, the failure could affect others in the environment within the big rooms. One of the largest risks is **fault propagation** since one failure can lead to others cascading onwards. For example, if one tenant suddenly is pulling large data, there is a resource exhaustion or a service failure, it impacts other tenants or clusters through memory, CPU, or bandwidth. In the absence of quarantine, some of such failures may go viral and affect the whole system negatively.

Another serious problem is the slow degradation of the car's **performance**. In situations when isolation fails or when it can only be partially applied, the dysfunction of one tenant negatively affects other tenants in terms of slow response times, lower availability, or, in the worst-case scenario, total blackouts. For instance, if one tenant spends much time on many requests because of a failed application, other clients may also run slowly or fail because of a lack of sufficient resources. As a result, there is a degradation of the services that are being offered as well as a general deterioration of the reliability of the cloud platform [3].

Different processes arise sharing a commodity, and this gives rise to the issue of **resource sharing**. To view one instance, when resources like CPU, storage or bandwidth are for example shared between several tenants these indicate that the malicious attempts of one tenant in one of the applications can cause resource contention, whereby many tenants fight for scarce resources. This can worsen the initial failure, and lead to other problems in terms of performance and accessibility of premises and resources for other tenants.

Lastly, another drawback of conventional isolation methods is the issue of **scalability**. Regular scale-up of the existing isolation strategies as the cloud environment size and dependency evolve is challenging. Such methods as virtualizing or using containers at a small scale may face challenges in handling large dynamic systems' workload. As the number of tenants and clusters increases, basic physical isolation techniques might become a bottleneck, which will not allow to achieve the desired level of high availability and performance of the system.

These issues prove that proper isolation techniques should be developed to address the difficulties of modern large-scale multi-tenant multi-cluster environments.

Isolation Techniques and Approaches

As the systems get bigger with multiple tenants and multiple clusters in place, the failures start cascading and destabilize the complete system. To avoid such occurrences, various isolation measures are taken so as to guarantee that such failures do not spread to other tenants or clusters. These techniques can be classified into five main types; logical isolation, physical isolation, resource isolation, fault isolation, and network isolation.

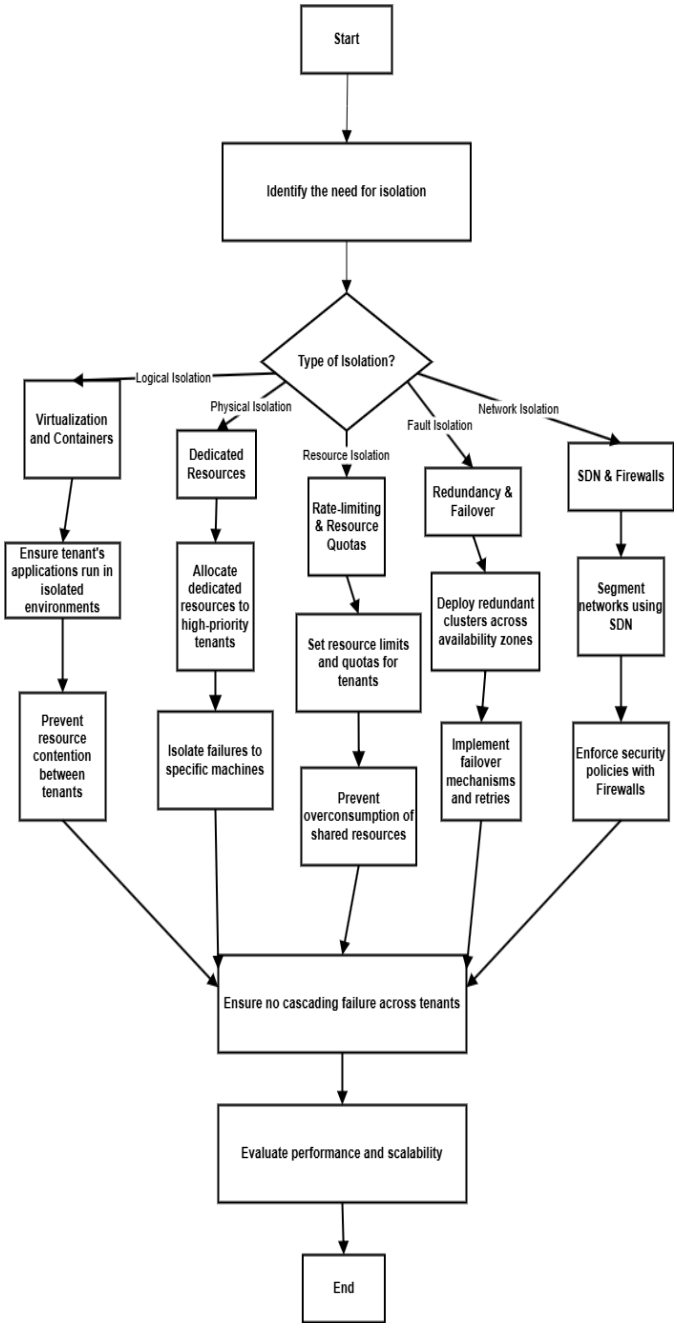

Fig 2: Flowchart diagram of Isolation Techniques and Approaches in a multi-tenant, multi-cluster cloud architecture

Logical Isolation

Virtualization and containerization are at the core the concept of logical isolation. Tenants are isolated through virtual Weber to run different operating systems on the same set of hardware physical resources but operate differently. This is implemented at the application level where each tenant's application is run in different containers such as Docker while on the same host and share the same operating system kernel [4].

Other forms of logical isolation comprise Linux and Kubernetes namespaces that also provide individual running spaces for processes, networks, and the file system [4]. To emphasize this difference in Kubernetes, tenants are given different namespaces that make them unique and isolated from the others.

Pseudocode example for Containerization:

```
docker run -d --name tenant1_container --memory=1GB --cpus=1.0 my_image
```

This pseudocode defines a space for developing programs for a certain tenant, reducing memory and CPU usage which can hinder other tenants from using the system.

Physical Isolation

Physical separation as physical separation means that the tenant or cluster has its own servers, storage, or network infrastructure with no shared resources with other tenants or clusters. Servers which are integrated and implemented to act in a bare-metal manner provide tenants with isolated hardware to ensure that failures affecting one tenant, will not affect others. This helps to make sure that failure of hardware or contention of some resource in one tenant environment cannot affect others [5].

Pseudocode for Physical Isolation Example:

```
allocate_server --tenant="Tenant1" --hardware="dedicated-server"
```

Resource Isolation

To avoid conflict of resources, **resource throttling** measures like **rate-limiting** and **quotas** are used. **ResourceQuotas**, in Kubernetes, prevents a tenant from utilizing a specific amount of CPU, memory or bandwidth more than the other tenants so that no tenant deprives the others of the shared resources. Resources can also be reserved for tenants, for example particular cores or amount of RAM which also reduces resource conflicts.

Pseudocode for Resource Quota Allocation:

```
apiVersion: v1
kind: ResourceQuota
 metadata:
   name: tenant1-quota
spec:
  hard:
   cpu: "4"
   memory: "8Gi"
```

This is a YAML configuration that manages resource quota for a tenant and the CPU and the memory in a cluster.

Fault Isolation

Fault isolation is employed to guarantee that faults cannot spread throughout the whole of the domain area. Redundancy, failover mechanisms and retry logic are particular measures that are useful when the failure has occurred. For instance, placement of clusters in availability zones minimize the effect of failure in one region than in the other. Elliptical process avoids system failure by ensuring it works albeit under diminished capability [6].

Pseudocode for Fault Tolerance (Retries):

```
def request_with_retry(url, retries=3):
    for attempt in range(retries):
        try:
            response = requests.get(url)
            return response
        except Exception as e:
            if attempt == retries - 1:
                raise e  # Raise error after final retry
            else:
                continue
```

This Python pseudocode shows a retry strategy, that called requests will be issued up to 3 times before an error is thrown.

Network Isolation

Network isolation aids in guaranteeing a secure form of communication that can be used among different tenants or clusters. SDN technology allows network administrators to control traffic flow between the tenants with ease. This lessens the probability of having a network problem that will span across several tenants. Firewalls and networking policies prevent unauthorized access and also help to separate different forms of communications [4].

Pseudocode for Network Isolation (SDN Example):

```
sdn create_network --tenant="Tenant1" --network="tenant1-net" --isolation=true
```

With this command, a network for Tenant 1 is created to be separate from the other tenants' network.

Case Study: Isolation Techniques in a Multi-Tenant, Multi-Cluster Environment for Cloud Service Providers

A case in point with a leading CSP who offers multi-cluster infrastructure where different tenants with different workloads share infrastructure [7]. It also showed tenancy issues such as cascading failures where problems with one tenant's applications affected the others due to performance and stability. To eliminate these risks the provider adopted several measures of isolation as a way of enhancing the system reliability.

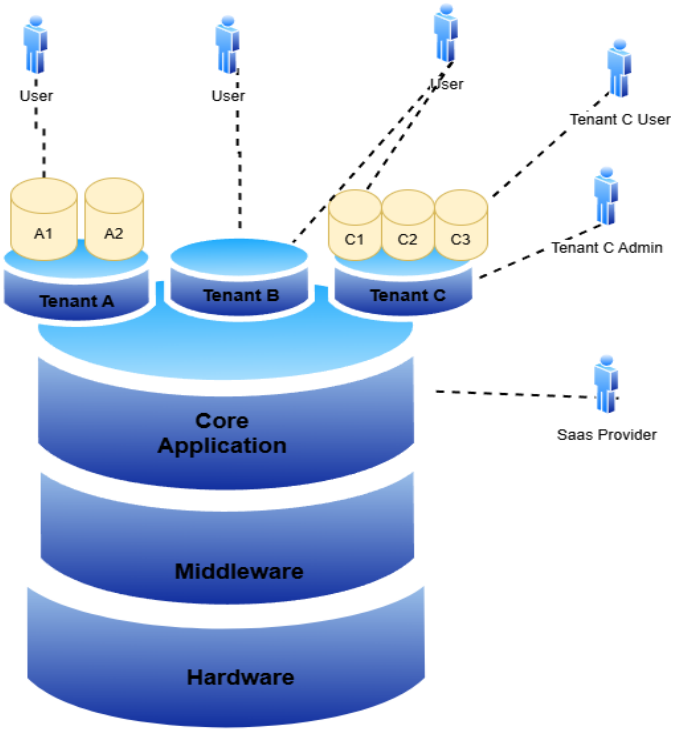

Fig 3: Multi-Tenant Application-Software Architecture

Initial Problem

Another attribute of the CSP's infrastructure was several clusters placed in diverse geographical zones; the tenants had to share CPU, memory, and network bandwidth [8]. One recurring problem was if there was some event in the tenant application, for example a memory leak, or increased traffic, which consumed even more resources. This could result in resource depletion leading to poor performance throughout the system and inconvenience to other tenants. All these cascading failures affected the overall service availability and reliability against faults.

Solution: Application of Isolation Techniques

The CSP adopted a multipronged strategy of isolation The challenges addressed logical isolation, physical isolation, resource isolation, fault isolation and network isolation. Containers mark the logical isolation; every tenant's application was housed in a unique Kubernetes namespace. This eliminated scenarios where issues in a given tenant's environment came to affect others and because resources were properly isolated, the required maximum consumption of resources was accommodated [9]. For high-priority tenants, isolation was achieved by allocating bare-metal servers, guaranteeing that a tenant could not monopolize the tenant's hardware while also preventing common-mode failure with other tenants.

For this purpose, the CSP used Kubernetes ResourceQuotas that made it possible to restrict the amount of CPU and memory consumed per namespace and then fairly distribute them. Request rates were also limited in order to minimize DoS attacks and overloads in requests. A work of fault isolation was done where more than one cluster was established over multiple availability zones in addition to fail over and retry logic to reduce unplanned interruptions. APIs pioneered circuit breakers to contain faults and shut them from influencing the system [10].

SDN was used to implement network isolation through provisioning of isolated virtual networks to each tenant. Firewalls were implemented for security policies to be set to secure the infrastructure from such threats or from collapsing for all tenants to Kiem. This multiple level of isolation concept highly improved system reliability and its security [11].

Results and Improvements

The application of these isolation techniques led to considerable enhancements in the behavior of the system. Resource controls depressed resource sharing while the separation of logical and physical spaces allowed for more consistent use of resources across the tenancy. Fault tolerance was improved through the use of redundant clusters and fail over, where system failure could not cause significant system downtime [12].

The network isolation also improved security; this meant that all the tenant communications were strictly off the clinical network. Following these modifications, the CSP addressed resource contention induced service **disruptions by 40%** and **system outages** caused by **cascading failures by 30%.**

Analysis and Evaluation of Isolation Techniques

Specific measures by the CSP controlling multi-tenants, multi-cluster setting successfully resolved issues related to cascading failures and resource rivalry. The four techniques were analyzed and compared in terms of scalability, fault tolerance, and implementation issues [13].

Comparison of Techniques

Based on the primary criterion of scalability, all of the isolation techniques used by CSPs offered a range of benefits and different disadvantages regarding fault tolerance and ease of implementation. Logical isolation through virtualization and containers, particularly by Kubernetes namespaces, was highly scalable, and fault-tolerant. Containers enabled simple addition of tenants with a segmented environment and isolation but container management as well as resource allocation were challenging in terms of the operational overhead. More physical isolation with dedicated resources means higher tolerance to faults and higher security but could not scale well because each tenant required new bare-metal servers which made it expensive and difficult to manage [14].

Resource limit by using Kubernetes ResourceQuotas and rate limiting served as a fair approach to establish resource fairness and avoid network overload [15]. The technique proved very effective, technically simple to apply but the major disadvantage was that it needed to be perfectly calibrated to ensure that it did not put a drag on resource usage. Redundancy and retries, and failover and multiple availability zones succeeded in reducing service downtime and maintaining service availability but were costly and less scalable. Again, the use of Software-Defined Networking (SDN) and firewalls allowed for good levels of network separation and security but were difficult to manage and added complexity to the use of this part of the network. The given techniques helped enhance the system reliability, security, and functionality given different environmental conditions [16].

Table 1: comparison of isolation techniques and their performance evaluation based on the analysis

Isolation Technique	Advantages	Challenges	Scalability	Fault Tolerance	Ease of Implementation
Logical Isolation (Virtualization & Containers)	High scalability with easy addition of tenants, effective isolation via namespaces, fault-tolerant	Requires significant expertise, potential overhead with resource limits	High	High	Medium
Physical Isolation (Dedicated Resources)	Highest fault tolerance and security, robust isolation between tenants	Less scalable, costly, harder to implement for many tenants	Low	Very High	Low
Resource Isolation (Quotas & Rate-Limiting)	Balances scalability and fault tolerance, easy to implement in Kubernetes	Requires fine-tuning, potential for underutilization or excessive throttling	Medium	Medium	High
Fault Isolation (Redundancy & Retries)	High fault tolerance, redundancy ensures uptime and resiliency	Adds infrastructure complexity, requires additional resources	Medium	Very High	Low
Network Isolation (SDN & Firewalls)	Robust security, scalable, isolates tenant traffic effectively	Complex management, requires specialized knowledge	High	High	Medium

Performance Evaluation

The application of these isolation techniques led to noticeable improvements in system performance:
System Uptime: Redundancy was applied throughout the system, further utilizing failover, which, in turn, decreased the downtime 30%.
Response Time: Division into address domains and resource domains reduced the load and enhanced effective response by avoiding interference between multiple requests using the same resource.
Resource Utilization: Resource quotas and rate limiting made balanced usage and distribution of resources to several applications more efficient and minimized the forms of wastage of more resources.

Table 2: Performance Evaluation

Key Performance Metric	Impact of Isolation Techniques
System Uptime	30% reduction in downtime due to redundancy and failover mechanisms.
Response Time	Reduced contention for resources through logical and resource isolation, improving response time.
Resource Utilization	Resource quotas and rate-limiting ensured efficient use of resources without over-provisioning.

Conclusion

Logical, physical, resource, fault, network isolation approaches have been evaluated and proven to minimize failure domino effect and resource competition in multi-tenancy and multi-cluster environments. Tools like the container, Kubernetes namespaces, and the SDN are deployed to enhance scalability, fault tolerance and security. However, many more questions like, more complex, Scalability issues, Higher costs for physical isolation still persist. A research direction for the future is the improvement of the integration process of the isolation management to increase scalability and efficiency of large complex systems. Furthermore, there is potential in the research on measures that incorporate aspects of various isolation concepts to improve system robustness, performance, and adaptability in large-scale cloud settings.

References

1. Padmapriya Duraisamy *et al.*, "Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale," Mar. 2023, doi: https://doi.org/10.1145/3582016.3582031.
2. Cartocci, N., Napolitano, M.R., Costante, G. and Fravolini, M.L., 2021. A comprehensive case study of data-driven methods for robust aircraft sensor fault isolation. *Sensors*, *21*(5), p.1645. https://www.mdpi.com/1424-8220/21/5/1645
3. Cho, J.H., Okuma, A., Sofjan, K., Lee, S., Collins, J.J. and Wong, W.W., 2021. Engineering advanced logic and distributed computing in human CAR immune cells. *Nature communications*, *12*(1), p.792. https://www.nature.com/articles/s41467-021-21078-7
4. Christiansen, J., Qualter, P., Friis, K., Pedersen, S.S., Lund, R., Andersen, C.M., Bekker-Jeppesen, M. and Lasgaard, M., 2021. Associations of loneliness and social isolation with physical and mental health among adolescents and young adults. *Perspectives in public health*, *141*(4), pp.226-236. https://journals.sagepub.com/doi/abs/10.1177/17579139211016077
5. Duraisamy, P., Xu, W., Hare, S., Rajwar, R., Culler, D., Xu, Z., Fan, J., Kennelly, C., McCloskey, B., Mijailovic, D. and Morris, B., 2023, March. Towards an adaptable systems architecture for memory tiering at warehouse-scale. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (pp. 727-741). https://dl.acm.org/doi/abs/10.1145/3582016.3582031

6. Holt-Lunstad, J. and Steptoe, A., 2022. Social isolation: An underappreciated determinant of physical health. *Current opinion in psychology*, *43*, pp.232-237. https://www.sciencedirect.com/science/article/pii/S2352250X21001111
7. Kang, Y., Yao, L. and Wang, H., 2021. Fault isolation and fault-tolerant control for Takagi–Sugeno fuzzy time-varying delay stochastic distribution systems. *IEEE Transactions on Fuzzy Systems*, *30*(4), pp.1185-1195. https://ieeexplore.ieee.org/abstract/document/9332248/
8. Liu, X., Du, J. and Ye, Z.S., 2021. A condition monitoring and fault isolation system for wind turbine based on SCADA data. *IEEE Transactions on Industrial Informatics*, *18*(2), pp.986-995. https://ieeexplore.ieee.org/abstract/document/9415155/
9. Noman, A. and Cheng, X., 2022. Bengali Isolated Speech Recognition Using Artificial Neural Network. In *Mechatronics and Automation Technology* (pp. 14-23). IOS Press. https://ebooks.iospress.nl/doi/10.3233/ATDE221144
10. Sabaghian, K., Khamforoosh, K. and Ghaderzadeh, A., 2023. Data Replication and Placement Strategies in Distributed Systems: A State of the Art Survey. *Wireless Personal Communications*, *129*(4), pp.2419-2453. https://link.springer.com/article/10.1007/s11277-023-10240-7
11. Şenel, B.C., Mouchet, M., Cappos, J., Friedman, T., Fourmaux, O. and McGeer, R., 2023. Multitenant containers as a service (CAAS) for clouds and edge clouds. *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/10365152/
12. Simić, M., Dedeić, J., Stojkov, M. and Prokić, I., 2024. A Hierarchical Namespace Approach for Multi-Tenancy in Distributed Clouds. *IEEE Access*. https://ieeexplore.ieee.org/abstract/document/10443611/
13. Van Zoonen, W. and Sivunen, A.E., 2022. The impact of remote work and mediated communication frequency on isolation and psychological distress. *European Journal of Work and Organizational Psychology*, *31*(4), pp.610-621. https://www.tandfonline.com/doi/abs/10.1080/1359432X.2021.2002299
14. Wang, D., Li, S., Zhao, Z., Zhang, X. and Tan, W., 2021. Engineering a second-order DNA logic-gated nanorobot to sense and release on live cell membranes for multiplexed diagnosis and synergistic therapy. *Angewandte Chemie International Edition*, *60*(29), pp.15816-15820. https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.202103993
15. Wijethilaka, S. and Liyanage, M., 2021. Survey on network slicing for Internet of Things realization in 5G networks. *IEEE Communications Surveys & Tutorials*, *23*(2), pp.957-994. https://ieeexplore.ieee.org/abstract/document/9382385/
16. Zhang, Y., Yasaei, R., Chen, H., Li, Z. and Al Faruque, M.A., 2021. Stealing neural network structure through remote FPGA side-channel analysis. *IEEE Transactions on Information Forensics and Security*, *16*, pp.4377-4388. https://ieeexplore.ieee.org/abstract/document/9517289/