Scholastic Seed Inc.

www.gjeis.com

global Journal of ENTERPRISE INFORMATION SYSTEM

Designing for Uptime: A Framework for High Availability Systems

– Sameer S Paradkar*

Chief Architect – Modernization, NTT Data, Mumbai

sameer.paradkar@global.ntt *https://orcid.org/0000-0001-7517-1574*

ARTICLE HISTORY

Paper Nomenclature: Case Based Study (CBS)

Paper Code: GJEISV16I3JS2024CBS2

Submission at Portal (www.gjeis.com): 07-July-2024

Manuscript Acknowledged:19-July-2024

Originality Check: 27-July-2024

Originality Test (Plag) Ratio (Plagiarism Checker X): 0%

Author Revert with Rectified Copy: 01-Aug-2024

Peer Reviewers Comment (Open): 04-Aug-2024

Single Blind Reviewers Explanation: 19-Aug- 2024

Double Blind Reviewers Interpretation: 24-Aug-2024

Triple Blind Reviewers Annotations: 02-Sept-2024

Author Update (w.r.t. correction, suggestion & observation): 07- Sept-2024

Camera-Ready-Copy: 19-Sept-2024

Editorial Board Excerpt & Citation: 25-Sept-2024

Published Online First: 30-Sept-2024

ABSTRACT

Purpose: The purpose of this study is to propose a comprehensive framework for designing high availability (HA) systems that meet diverse operational demands. By focusing on Non-Functional Requirements (NFRs) as foundational elements, this research addresses how HA can be achieved while balancing scalability, reliability, and performance. The framework is aimed at assisting system architects in navigating the complexities of designing resilient, enterprise-grade systems capable of meeting modern uptime requirements, from three nines (99.9%) to six nines (99.9999%).

Design/Methodology/Approach: This study employs a mixed-method approach combining theoretical modelling, case analysis, and scenario-based evaluations. The methodology includes fault tree analysis, reliability block diagrams, and simulation testing to model and validate the impact of NFRs on system performance and availability. Insights from real-world implementations of HA systems are incorporated to contextualize the proposed framework. Tools and techniques like distributed systems, global load balancing, and multi-region orchestration are critically assessed.

Findings: The research identifies key methodologies and technologies for achieving high availability across system tiers, including the UI, application, database, and integration tiers. It highlights the critical interplay between HA and other software quality attributes such as security, maintainability, and performance. Findings suggest that incorporating advanced orchestration techniques and predictive monitoring significantly enhances the reliability and scalability of HA systems. The study also underscores the importance of iterative feedback and real-world testing in aligning system design with operational goals.

Originality Value: This paper contributes to the field by providing a structured High Availability System Design Matrix that aligns NFRs with availability KPIs and specific architectural solutions. By addressing the practical challenges of designing highly available systems, the study bridges the gap between theoretical NFR modeling and real-world system design. Its insights into cost-performance trade-offs and the integration of HA with scalability provide valuable guidance for enterprises and system architects..

Paper Type: Case Based Study

KEYWORDS: Non-Functional Requirements | Highly Available Systems | Software Attributes | Key Performance Indicators

**Corresponding Author (Sameer)*

- Present Volume & Issue (Cycle): Volume 16 | Issue-3 | Jul-Sept 2024
- International Standard Serial Number: Online ISSN: 0975-1432 | Print ISSN: 0975-153X
- DOI (Crossref, USA) https://doi.org/10.18311/gjeis/2024
- Bibliographic database: OCLC Number (WorldCat): 988732114
- Impact Factor: 3.57 (2019-2020) & 1.0 (2020-2021) [CiteFactor]
- Editor-in-Chief: Dr. Subodh Kesharwani
- Frequency: Quarterly
- Published Since: 2009
- Research database: EBSCO https://www.ebsco.com
- Review Pedagogy: Single Blind Review/ Double Blind Review/ Triple Blind Review/ Open Review

- Publishers: Scholastic Seed Inc. and KARAM Society
- Place: New Delhi, India.
- Repository (figshare): 704442/13

Introduction

In an increasingly digital world, where applications are central to both enterprise operations and user experiences, ensuring continuous availability has become a non-negotiable requirement. High Availability (HA) systems, designed to minimize downtime and ensure reliable service delivery, form the backbone of resilient infrastructure. From e-commerce platforms to critical healthcare systems, the ability to maintain uptime under varied operational challenges directly influences organizational success.

Non-Functional Requirements (NFRs) play a pivotal role in shaping HA system designs. Unlike functional requirements that address specific system behaviors, NFRs focus on overarching attributes such as availability, reliability, maintainability, and disaster recovery. These attributes guide architectural decisions, determine technology stacks, and influence the methodologies adopted in system design.

This paper introduces a structured framework for HA system design, built on a matrix aligning availability Key Performance Indicators (KPIs) with design methodologies and technologies. The framework encompasses diverse operational scenarios, ranging from three nines (99.9%) to six nines (99.9999%) of uptime, addressing the unique demands of each tier: UI, application, database, and integration.

The objectives of this study are threefold:

- To demonstrate how NFRs serve as enablers of system performance and resilience rather than constraints.
- To provide actionable insights into modelling high availability using techniques such as fault tree analysis, reliability block diagrams, and simulation testing.
- To propose a practical design matrix that balances cost, complexity, and operational demands.

By presenting a holistic approach to HA system design, this research aims to empower system architects with the tools and methodologies necessary for creating robust, scalable, and efficient infrastructures. Through this framework, organizations can align their technical strategies with business-critical uptime requirements, ensuring competitive advantage in an ever-evolving technological landscape.

Key Performance Indicators - KPIs for Highly Available Systems

Identifying and measuring the right Key Performance Indicators (KPIs) is fundamental in the design and maintenance of highly available systems. This section introduces the most critical KPIs, such as uptime/downtime metrics, RTO (Recovery Time Objective) and RPO (Recovery Point Objective), and system reliability indices. These KPIs provide tangible metrics to assess the performance and reliability of systems.

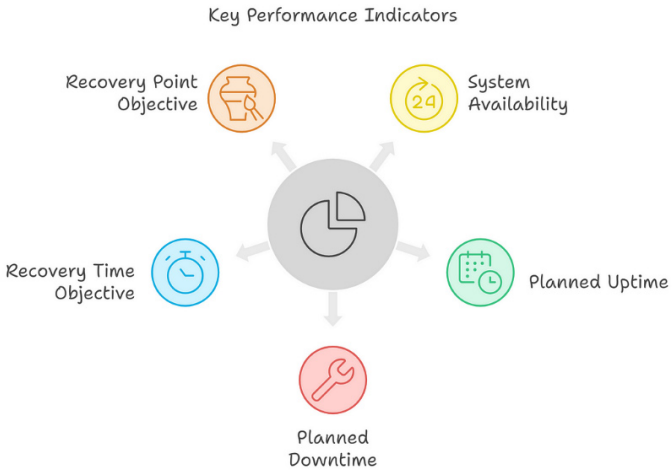

Figure:1 KPIs Highly Available Systems

Key Performance Indicators for Highly Available Systems

- **Availability** measures the proportion of time a system is functional and available. Examples:
  - *System uptime percentage: 99.99% (Four nines)*
  - *Maximum unplanned downtime: 52.56 minutes/year*
  - *Service availability during business hours: 99.999%*
  - *Failure rate: < 0.01% of total transactions*
- **Planned Uptime** refers to the time a system is expected to be operational and excludes planned maintenance periods. Examples:
  - *Business hours coverage: 24/7 (or specified hours)*
  - *System operational hours: e.g., 8640 hours/year (accounting for maintenance)*
  - *Peak period availability: 100% during specified critical times*
  - *Scheduled operation time: 99.9% adherence to planned schedule*
- **Planned Downtime** accounts for periods when the system is deliberately taken offline for maintenance. Examples:
  - *Maintenance window duration: 4 hours/month*
  - *Maximum maintenance frequency: Once per month*
  - *Off-peak maintenance timing: e.g., 2 AM - 6 AM UTC*
  - *Advance notification period: 7 days before planned downtime*

- **RTO** (Recovery Time Objective) gauges the time taken to recover from a failure and resume operations. Examples:
  - *Critical systems recovery: < 15 minutes*
  - *Non-critical systems recovery: < 4 hours*
  - *Full system restoration: < 8 hours*
  - *Service level restoration time: < 30 minutes*
- **RPO** (Recovery Point Objective) assesses the maximum acceptable amount of data loss measured in time. Examples:
  - *Critical data loss tolerance: < 5 minutes*
  - *Transaction data loss limit: < 30 seconds*
  - *Backup frequency: Every 15 minutes*
  - *Data sync delay tolerance: < 1 minute*

Implementing and Monitoring KPIs in High Availability System Design:

The implementation of KPIs is not a one-time activity but a continuous process requiring consistent monitoring and adjustment. This subsection will explore strategies for implementing these KPIs within the lifecycle of system design, from initial planning to operational maintenance. The role of automated monitoring tools and incident response mechanisms in tracking these KPIs will be examined. By continuously monitoring these metrics, organizations can proactively manage and maintain the high availability of their systems, ensuring they meet the required performance standards.

Modelling High Availability: Non-Functional Requirements (NFRs)

Non-Functional Requirements (NFRs) play a pivotal role in the design of highly available systems. This section will cover the specific NFRs that are crucial for high availability, such as availability, reliability, maintainability, and disaster recovery. We will explore how these NFRs guide architectural decisions and the selection of technologies and methodologies in system design. The discussion will include insights into how NFRs are not just constraints but enablers that drive the performance and resilience of a system.

Techniques for Modelling High Availability Requirements:

Modelling high availability involves a range of techniques and methodologies. This subsection focuses on effective approaches to model NFRs for high availability, including the use of scenario-based techniques, fault tree analysis, and reliability block diagrams. These techniques help in visualizing and quantifying the impact of different design choices on system availability and reliability. We will also discuss the role of simulation and testing in validating NFRs and ensuring that the system meets the high availability requirements outlined in the design phase.

Modelling high availability involves a range of techniques to ensure systems can handle failures and maintain uptime. The following approaches help visualize, quantify, and validate the impact of different design choices on availability and reliability

Table1: Impact of different design choices on availability and reliability

Technique	Description	Application	Example	Key Metric	Challenge
Scenario Testing	Models' system behavior during failures like hardware crashes or traffic spikes.	Ensures systems meet real-world operational needs under stress.	Simulating a data center outage on Black Friday.	Recovery Objective (RPO)	Needs accurate scenario design.
Fault Tree Analysis	Identifies root causes of potential failures.	Pinpoints weak points and suggests fixes.	Clustering to mitigate database outages.	Failure Time (MTBF)	Requires domain expertise.
Reliability Diagrams	Visualizes system reliability and dependencies.	Plans redundancies to minimize failure impact.	Redundant services in microservices setup.	Failure Time (MTTF)	Complexity in large systems.
Simulation & Testing	Mimics failures to test resilience.	Validates system tolerance for real-world conditions.	Chaos Monkey disrupting servers to test faults.	Downtime Percentage	Resource-intensive for large scale.
Quantitative Models	Predicts system reliability with statistical methods.	Aids precise planning for uptime.	Server failure probability analysis.	Uptime Percentage	Relies on accurate data.
Iterative Feedback	Continuously updates system designs.	Aligns designs with real-world performance insights.	Improving disaster recovery after outages.	None	Depends on monitoring accuracy.
Redundancy Setup	Ensures failover capability with backup systems.	Minimizes downtime with active-active or active-passive models.	Dual data center with auto-failover.	Failover Time	High cost and complexity.
SLAs Integration	Sets HA objectives like RPO and RTO for business needs.	Aligns system design with uptime guarantees.	1-hour RTO for financial apps.	RPO, RTO	Misaligned goals create gaps.

Incorporating Scalability into Broader System Requirements:

While focusing on high availability, it is crucial to ensure that scalability is not compromised. High availability and scalability must be harmoniously integrated to meet the growing demands of modern applications. Here's how scalability can be incorporated into broader system requirements without affecting availability:

- **Elastic Resource Allocation**: Dynamically allocate resources to handle traffic spikes while maintaining system uptime. For example, cloud platforms like **AWS Auto Scaling** and **Kubernetes Horizontal Pod Autoscaler** adjust resources in real time to ensure both scalability and availability.
- **Scalable Database Design**: Utilize database techniques like **sharding** and **replication** to handle large data volumes while ensuring that a single failure does not impact the entire system.
- **Adaptable Load-Balancing**: Implement load-balancing strategies that distribute traffic evenly across multiple instances, ensuring that the system remains responsive even as traffic increases. Tools like **NGINX**, **HAProxy**, and **AWS Elastic Load Balancer** are key enablers of both scalability and availability.

By integrating these strategies, systems can scale effortlessly without compromising the high availability required for business-critical applications.

Modelling high availability through these techniques provides valuable insights into how systems respond to failures, ensuring that they remain operational and reliable. High availability NFRs, when aligned with scalability requirements, enable the design of systems that not only meet uptime targets but also scale with user demand. Continuous monitoring, testing, and iteration are key to maintaining high availability and adapting to evolving system needs.

High Availability System Design Framework

The table provides a comprehensive breakdown of availability Key Performance Indicators (KPIs), illustrating the relationship between system uptime goals and the corresponding methodologies, techniques, and architectural solutions required to achieve these targets. It spans availability levels from 99.9% ("Three Nines") to 99.9999% ("Six Nines"), highlighting the increasing sophistication and complexity of strategies as the downtime tolerance diminishes. Each availability tier is mapped to tailored approaches across critical system tiers: UI, application, database, and integration. For example, higher availability levels demand advanced multi-region deployments, distributed databases, and active-active architectures, coupled with real-time monitoring and orchestration. This holistic perspective underscores how incremental improvements in availability necessitate strategic investments in redundancy, fault tolerance, and failover mechanisms across the entire system stack, ensuring resilience and reliability for modern, mission-critical applications.

Table 2: High Availability System Design Framework

Availability KPI	Downtime	Methodology	Technique	UI Tier	Application Tier	Database Tier	Integration Tier
99.9% (Three Nines)	~8.76 hours/year downtime	Single-region failover and basic redundancy	Load balancers (e.g., AWS ELB), regular backups, monitoring tools (e.g., Nagios, Zabbix)	Caching and CDN for static assets	Load-balanced application servers	Primary-replica database setup, basic failover	Basic messaging queues (e.g., RabbitMQ), periodic health checks for failover.
99.95%	~4.38 hours/year downtime	Multi-zone failover and fault tolerance	Region-aware load balancers, automated backup restoration	Advanced CDN and progressive loading	Multi-zone active-passive failover	Multi-AZ database clusters (e.g., AWS RDS Multi-AZ)	Redundant messaging brokers with automated failover.
99.99% (Four Nines)	~52.6 minutes/year downtime	Multi-region deployment and distributed systems	Global load balancing (e.g., AWS Route 53), real-time monitoring (e.g., Datadog)	Real-time UI monitoring	Auto-scaling and container orchestration (e.g., Kubernetes)	Database sharding and read replicas (e.g., PostgreSQL, MySQL)	High-availability integration brokers (e.g., Kafka clusters).
99.999% (Five Nines)	~5.26 minutes/year downtime	Active-active architecture and fault-tolerant systems	Early failure detection, failover, load balancing, geographic distribution, cluster technology, First Hop Redundancy Protocol (FHRP), Redis Enterprise	A/B testing for UI updates	Stateless app servers with session replication	Distributed databases with automated failover (e.g., Cassandra, MongoDB)	Guaranteed message delivery queues (e.g., Kafka with exactly-once semantics).
99.9999% (Six Nines)	~31.5 seconds/year downtime	Geographically distributed active-active architecture	Advanced failover orchestration (e.g., Istio, Consul), multi-region consensus (e.g., Raft, Paxos)	Global CDN with edge caching	Zero-downtime multi-region app orchestration	Strong consistency distributed databases (e.g., Google Spanner, Cockroach DB)	Real-time failover across hybrid/multi-cloud integrations.

Interplay of HA with Key Software Quality Attributes

In the realm of systems striving for high availability (HA), it's essential to recognize the complex interplay and interdependency between HA and other critical software quality attributes. High availability is not an isolated characteristic but is deeply intertwined with attributes like security, maintainability, performance, resilience, and scalability. This interdependency is crucial: optimizing for high availability must be carefully balanced with these attributes to ensure overall system robustness and efficiency. Here, we examine how high availability interacts with these key attributes, emphasizing the importance of a balanced approach for the development of comprehensive software systems.

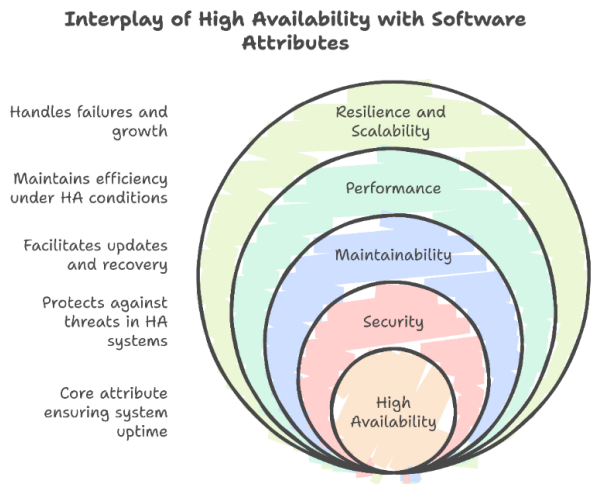

Figure 2: Interplay of High Availability with Key Software Attributes

Interplay of High Availability with Key Software Attributes

- **Security**: The relationship between high availability and security is intricate. As systems are designed for higher availability, they often become more vulnerable to security threats due to their continuous operational state and extensive redundancy mechanisms. Effective HA strategies need to be aligned with robust security measures, ensuring that the system's continuous availability doesn't become an exploitable weakness. Security mechanisms should be capable of protecting the system without impeding its availability.
- **Maintainability**: High availability and maintainability have a symbiotic relationship. An HA system needs to be maintainable to adapt efficiently to various scenarios that might otherwise impact availability. Conversely, a maintainable system can more easily incorporate enhancements for availability. High maintainability facilitates quick recovery from failures and seamless updates, which are essential for uninterrupted service.
- **Performance**: The interdependency between high availability and performance is pivotal. An HA system must maintain or enhance its performance, despite the complexity added by redundancy and failover mechanisms. This requires a balance where redundancy does not adversely affect system performance. The challenge lies in ensuring that the system maintains high performance under different operational conditions without compromising availability.
- **Resilience**: High availability directly impacts resilience. As systems are designed for higher availability, they must also enhance their resilience strategies to handle a broader range of potential failures. HA systems need to develop sophisticated redundancy and fault tolerance to accommodate diverse operational challenges. The goal is to ensure system stability and reliable functioning, even under adverse conditions.
- **Scalability**: The interplay between high availability and scalability is critical. HA systems must be designed to maintain availability even as they scale. This involves designing for scalable redundancy, where the system can handle increased loads and growth without sacrificing availability. Strategies for dynamic scaling are essential to maintain service continuity during varying demand levels.

Table 3: Interplay of High Availability with Key Software Attribute

Quality Attribute	Relationship with Availability	Visual Indicator	Actionable Steps
Resilience	Integral	(100%)	Implement redundant systems for uninterrupted service.
Performance	Balancing Act	(95%)	Minimize downtime while maximizing performance.
Security	Synergistic	(85%)	Ensure availability of secure channels during attacks.
Usability	Complementary	(90%)	Design intuitive systems that are always accessible.
Scalability	Interdependent	(80%)	Plan scalability to maintain availability under peak load.

In summary, the interplay and interdependency between high availability and other software quality attributes are foundational in designing sophisticated and efficient systems. Understanding and managing these relationships are essential for developing systems that are not only highly available but also secure, maintainable, high-performing, resilient, and scalable. This comprehensive approach is indispensable in the evolving landscape of system design and development.

Methodologies for Evaluating and Making Informed Trade-offs in High Availability System Design

Evaluating trade-offs in the design of High Availability (HA) systems requires a structured, methodical approach. By employing various methodologies, comprehensive insights can be gained, ensuring that high availability is achieved without compromising other critical aspects of the system. Let's explore these methodologies:

Table 4: Different Methodologies for evaluating and making informed Trade-offs

Methodology	Description	Purpose	Example/Application
Cost-Benefit Analysis	Quantifies costs (e.g., infrastructure) and benefits (e.g., uptime) to determine net gain or loss.	Helps understand the financial impact of HA decisions.	Evaluating whether redundant servers justify the cost for a critical application.
Scenario-Based Evaluation	Creates hypothetical failure and recovery scenarios to assess system robustness.	Provides practical insights into HA strategy performance in real-world conditions.	Simulating a database failure to analyze recovery strategies.
Performance Modelling	Uses simulations to predict HA impact on performance and availability metrics.	Identifies potential performance bottlenecks and ensures uptime during peak load conditions.	Simulating a spike in user traffic to test system recovery time.
Risk Assessment	Identifies risks (e.g., system failure, data loss) and develops mitigation strategies.	Ensures system integrity by proactively addressing potential vulnerabilities.	Analyzing the impact of power outages on critical services and planning failover.
Stakeholder Feedback	Gathers input from users, clients, and IT staff through surveys, interviews, or testing.	Aligns HA strategies with business and user needs.	Collecting user feedback to validate the effectiveness of failover mechanisms.

In conclusion, employing a combination of these methodologies enables a holistic and nuanced understanding of HA trade-offs. It empowers developers and architects to make well-informed decisions, ensuring that high availability is achieved in harmony with the system's overall functionality, performance, and user satisfaction. By carefully considering and managing these trade-offs, the design of HA systems becomes a balanced act of technical proficiency, economic pragmatism, and user-centricity.

Conclusion and Future Work

In this study, we have explored the intricacies of designing highly available systems, emphasizing the critical role of Non-Functional Requirements (NFRs) as the foundation for achieving operational resilience. The proposed High Availability (HA) framework, enriched by a structured design matrix and real-world methodologies, provides a comprehensive toolkit for architects and engineers to balance the often competing demands of availability, scalability, and performance.

By integrating key strategies such as fault tree analysis, reliability block diagrams, and predictive monitoring, this research bridges the gap between theoretical principles and practical implementation. The interplay of HA with other software quality attributes like security, maintainability, and scalability further highlights the necessity of a holistic approach. Real-world examples demonstrate how system architects can navigate complex trade-offs to deliver robust solutions capable of meeting modern uptime expectations, from three nines to six nines.

However, high availability is not a static goal; it is a continuous journey. As technology landscapes evolve, future work must address emerging challenges, such as the integration of HA with sustainability goals, the use of AI and machine learning for proactive system monitoring, and the growing reliance on hybrid and multi-cloud environments. The exploration of self-healing systems and advanced orchestration techniques will also be crucial to pushing the boundaries of system resilience.

Ultimately, the pursuit of high availability is a strategic investment in user trust, operational efficiency, and business continuity. By continuously iterating on the principles, practices, and tools discussed in this paper, organizations can build systems that not only meet but exceed the expectations of an increasingly digital and interconnected world.

References

- Bass, L., Clements, P., & Kazman, R. (2013). *Software architecture in practice* (3rd ed.). Addison-Wesley. (Chapter 5: Non-Functional Attributes in Software Architecture)
- Correia-Neto, J., Gonçalves, J., & Sales, T. (2019). Non-functional requirements in health information systems: A systematic mapping. *Journal of Software Engineering Research and Development, 7*(1), 1–25. https://doi.org/10.1186/s40411-019-0067-7
- Hassine, J. (2015). Describing and assessing availability requirements in the early stages of system development. *Software & Systems Modeling, 15*(2), 1–15. https://doi.org/10.1007/s10270-013-0382-0
- Kazman, R., Klein, M., & Clements, P. (2000). ATAM: Method for architecture evaluation. *Technical Report CMU/SEI-2000-TR-004*. Carnegie Mellon University. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5177
- Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (pp. 35–46). ACM. https://doi.org/10.1145/336512.336523
- Ouhbi, S., Idri, A., Luis Fernández-Alemán, J., & Toval, A. (2015). Requirements engineering in software product lines: A systematic mapping study. *Information and Software Technology, 57*(1), 52–66. https://doi.org/10.1016/j.infsof.2014.09.013
- Summers, J. (2020). Towards the formalization of non-functional requirements in engineering design. *Research in Engineering Design, 31*(3), 347–366. https://doi.org/10.1007/s00163-020-00345-6
- Sommerville, I., & Kotonya, G. (1998). Viewpoints for requirements definition in distributed systems. *Software Engineering Journal, 12*(1), 5–16. https://doi.org/10.1049/sej.1998.0002
- Tracing non-functional requirements. (2005). In *Engineering and Managing Software Requirements* (pp. 253–268). Springer. https://doi.org/10.1007/978-1-4471-2239-5_14
- Wan, J., Tang, S., Hua, Q., et al. (2015). Cloud computing for high-availability services. *IEEE Communications Magazine, 53* (3), 59–65. https://doi.org/10.1109/MCOM.2015.7060480

GJEIS Prevent Plagiarism in Publication

The Editorial Board had used the Checker X – anti-plagiarism software tool which is a fully-automatic machine learning text-recognition system made for detecting, preventing and handling plagiarism and trusted by thousands of institutions across worldwide. Checker X is GDPR compliant with privacy by design and an uptime of 99.9% and have trust to be the partner in academic integrity. https://plagiarismcheckerx.com/ tool to check the originality and further affixed the similarity index which is {0%} in this case (See below Annexure 16.3.6). Thus, the reviewers and editors are of view to find it suitable to publish in this Volume 16, Issue-3, Jul-Sept 2024.

Annexure 16.3.6

Submission Date	Submission	Word Count	Character Count
27-July-2024	Plagiarism Checker X	3095	23390

Analyzed Document	Submitter email	Submitted by	Similarity
3.2 CBS2_Sameer_GJEIS Jul to Sept 2024.docx	sameer.paradkar@global.ntt	Sameer S Paradkar	0%

Plagiarism Checker X - Report

Originality Assessment

Overall Similarity: **0%**

Reviewers Memorandum

Reviewer's Comment 1: The paper presents a well-structured and insightful discussion on designing high availability (HA) systems. The topic is highly relevant, given the increasing dependency on uninterrupted digital services across industries. The proposed framework effectively aligns Non-Functional Requirements (NFRs) with high availability objectives, offering practical insights for system architects. However, the paper could benefit from a more detailed discussion on real-world trade-offs, such as cost implications, performance overheads, and security concerns in achieving ultra-high availability. Addressing these aspects may enhance the study's practical applicability.

Reviewer's Comment 2: The research adopts a rigorous mixed-method approach, combining theoretical modeling, case analysis, and scenario-based evaluations. The use of fault tree analysis, reliability block diagrams, and simulation testing strengthens the methodology. One area that could be expanded is the comparative analysis with existing HA frameworks. A brief discussion on how the proposed framework differs from or improves upon current industry practices would enhance its academic and practical contribution.

Reviewer's Comment 3: The paper effectively highlights the interplay between HA and other software quality attributes, such as security, maintainability, and scalability. The discussion on Key Performance Indicators (KPIs) is well-articulated, providing clear metrics for assessing HA systems. However, incorporating more real-world examples or case studies—especially from industries like cloud computing, financial services, or e-commerce would make the findings even more impactful. Additionally, a discussion on emerging technologies like AI-driven predictive monitoring for HA systems could add a future-oriented perspective.

Citation

Sameer S Paradkar
"Designing for Uptime:
A Framework for High Availability Systems"
Volume-16, Issue-3, Jul-Sept 2024. (www.gjeis.com)

https://doi.org/10.18311/gjeis/2024
Volume-16, Issue-3, Jul-Sept 2024
Online iSSN : 0975-1432, **Print iSSN** : 0975-153X
Frequency : Quarterly, Published Since : 2009

Google Citations: Since 2009
H-Index = 96
i10-Index: 964

Source: https://scholar.google.co.in/citations?user=S47TtNkAAAAJ&hl=en

Conflict of Interest: Author of a Paper had no conflict neither financially nor academically.

Editorial Excerpt

The article has 0% of plagiarism which is the accepted percentage as per the norms and standards of the journal for publication. As per the editorial board's observations and blind reviewers' remarks the paper had some minor revisions which were communicated on a timely basis to the authors (Sameer), and accordingly, all the corrections had been incorporated as and when directed and required to do so. The comments related to this manuscript are noticeably related to the theme "Designing for Uptime: A Framework for High Availability Systems" both subject-wise and research-wise. The paper is well-written, structured logically, and maintains a professional tone. The abstract and introduction effectively set the stage for the discussion, and the conclusions are well-supported by evidence. The language is clear and accessible, making it suitable for both academic and practitioner audiences. Overall, the paper contributes valuable insights into the future of education. After comprehensive reviews and the editorial board's remarks, the manuscript has been categorized and decided to publish under the "**Case Based Study**" category.

Acknowledgement

The acknowledgment section is an essential part of all academic research papers. It provides appropriate recognition to all contributors for their hard work and effort taken while writing a paper. The data presented and analyzed in this paper by (Sameer) were collected first handily and wherever it has been taken the proper acknowledgment and endorsement depicts. The authors are highly indebted to others who facilitated accomplishing the research. Last but not least, endorse all reviewers and editors of GJEIS in publishing in the present issue.

Disclaimer

All views expressed in this paper are my/our own. Some of the content is taken from open-source websites & some are copyright free for the purpose of disseminating knowledge. Those some we/I had mentioned above in the references section and acknowledged/cited as when and where required. The author/s have cited their joint own work mostly, and tables/data from other referenced sources in this particular paper with the narrative & endorsement have been presented within quotes and reference at the bottom of the article accordingly & appropriately. Finally, some of the contents are taken or overlapped from open-source websites for knowledge purposes. Those some of i/we had mentioned above in the references section. On the other hand, opinions expressed in this paper are those of the author and do not reflect the views of the GJEIS. The authors have made every effort to ensure that the information in this paper is correct, any remaining errors and deficiencies are solely their responsibility.