International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 1587-1594
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Scaling Autonomous Decision Systems: A Framework for Organizational Deployment

Suganya Nagarajan*

Independent Researcher, USA
* **Corresponding Author Email:** suganya.n.contact@gmail.com- **ORCID:** 0000-0002-5247-7750

Article Info:

DOI: 10.22399/ijcesen.5097
Received : 02 February 2026
Revised : 20 March 2026
Accepted : 25 March 2026

Keywords

Autonomous Decision Systems,
Organizational Capability,
Execution Permission,
Control Plane Enforcement,
Microservices Architecture

Abstract:

Autonomous decision systems have become foundational infrastructure in large-scale digital environments, mediating customer interactions across retail, payments, and subscription services at throughputs that demand both organizational coherence and operational precision. Yet the predominant failure mode in these systems is not algorithmic, it is structural, arising from ambiguous ownership boundaries, duplicated safety controls, and inconsistent enforcement across heterogeneous product teams operating on shared customer-facing surfaces. This article presents a system-level framework that treats autonomy as an organizational capability rather than a property of individual services. The framework introduces a three-layer separation of responsibilities: decision intent, execution permission, and content selection distributed across product teams, a centralized autonomy platform, and presentation layers respectively. The autonomy platform is responsible for enforcing execution safety consistently across teams while remaining independent of business logic and presentation decisions.To support low-latency execution environments, the framework distinguishes between control-plane policy computation and runtime policy enforcement, allowing policy decisions to be computed asynchronously while enabling lightweight enforcement during request execution. The article also identifies scheduled execution as a significant source of latent risk and proposes runtime permission evaluation as a mechanism for maintaining safety under changing system conditions.

1. Introduction

Autonomous decision systems have evolved from isolated model deployments into foundational infrastructure for large-scale digital platforms. In domains such as online retail, digital payments, and subscription services, automated decision pipelines now generate millions of decisions daily, determining which promotions are shown, which risks are communicated, and which customer experiences are triggered in real time.

Despite significant advances in machine learning models and decision optimization techniques, many operational failures in autonomous systems arise not from incorrect decision logic but from structural challenges within the organizations that deploy them. Large enterprises typically operate through dozens of independent product teams contributing decision logic to shared customer-facing platforms. When the operational boundaries governing these decisions are unclear, teams independently implement safety controls, fallback logic, and override mechanisms. Over time, this leads to duplicated safeguards, inconsistent enforcement, and hidden dependencies across services. Similar patterns have been observed in production machine learning systems, where hidden technical debt accumulates across pipelines, interfaces, and configuration layers, often surfacing only during incidents or system stress conditions [2].

These structural challenges are further amplified in microservices-based architectures, which deliberately distribute service ownership to enable independent development and deployment. While this architectural style increases agility and scalability, it also increases coordination complexity when cross-cutting operational concerns—such as safety enforcement, rate limits, and system health constraints—are implemented independently across services. Research on microservices and distributed system architectures consistently identifies poorly defined service

boundaries as a major contributor to maintenance overhead and operational fragility in large-scale systems [3].In high-throughput decision environments, these organizational and architectural issues can directly affect system performance. Customer-facing presentation layers often operate under strict latency constraints measured in single-digit milliseconds. In concurrent execution environments, excessive synchronization and coordination between components can introduce contention that significantly reduces throughput and increases latency exposure. Concurrency research demonstrates that poorly designed synchronization mechanisms can severely limit scalability under high levels of parallelism [1].The challenge becomes even more pronounced as autonomous decision systems scale across multiple product teams operating on shared infrastructure. Each team may generate candidate decisions independently, but the execution of these decisions occurs within a shared operational environment subject to common constraints such as system health, operational readiness, and organizational risk tolerance. When enforcement mechanisms are implemented independently by each team, coordination during incidents becomes difficult and automation reliability decreases. Distributed systems research has long demonstrated that achieving agreement and coordination across independent components under failure conditions is inherently complex and computationally expensive [4].These challenges suggest that autonomy should not be treated as a property of individual services or decision models. Instead, it must be treated as an **organizational capability**, requiring explicit boundaries between the actors responsible for generating decisions, enforcing operational constraints, and presenting decisions to users.To address this challenge, this article introduces a system-level framework that separates autonomous decision systems into three distinct layers: **decision intent, execution permission, and content selection**. Product teams define decision intent by specifying candidate actions based on domain-specific signals and business objectives. A centralized autonomy platform governs execution permission by enforcing organization-wide safety constraints such as blast radius limits, operational readiness conditions, and fallback policies. Presentation layers perform content selection by ranking and exposing permitted actions to users according to contextual relevance and experimentation strategies.To support scalability in high-throughput environments, the proposed framework further distinguishes between **control-plane policy computation and runtime policy enforcement**. Policies governing autonomous execution are computed asynchronously using aggregate signals such as system health indicators, incident signals, and operational budgets. These policies are then distributed to runtime systems in advance, enabling constant-time enforcement during decision execution without introducing synchronous dependencies on the request path.The contributions of this work are threefold:

1. It conceptualizes autonomous decision systems as an **organizational capability** rather than a localized service feature.
2. It introduces a **three-layer architectural model** separating decision intent, execution permission, and content selection across organizational actors.
3. It presents an **operational enforcement architecture** that combines control-plane policy computation with lightweight runtime enforcement to support low-latency, high-throughput autonomous systems.

By formalizing these architectural and organizational patterns, this work provides practical guidance for organizations seeking to deploy autonomous decision systems reliably across complex enterprise environments.

2. Why Autonomy Fragments at Scale

As autonomous decision systems expand across multiple product teams, a predictable pattern of fragmentation emerges. Each team, responsible for its own product domain, independently implements the safeguards required to operate its decision logic—such as rate limits, fallback mechanisms, and override procedures. Over time, these safeguards proliferate across services, often in incompatible forms that are difficult to coordinate or enforce uniformly. Similar operational complexities have been widely documented in production machine learning systems, where hidden technical debt accumulates across pipelines, configuration layers, and infrastructure dependencies as systems evolve [2].Microservices-based architectures, the dominant deployment model for large-scale digital platforms, further amplify this fragmentation. These architectures intentionally distribute service ownership across independent teams to enable rapid development and deployment. However, this distribution also increases coordination complexity when cross-cutting operational controls are implemented independently within each service. Research on microservices and distributed system architectures consistently identifies poorly defined service

boundaries as a major contributor to operational complexity and long-term maintenance overhead [3].One of the most significant consequences of this fragmentation is the emergence of synchronous dependencies within critical request paths. When product teams embed safety checks directly within their runtime execution logic, coordination costs increase under concurrent load. Distributed systems research demonstrates that achieving coordination among independent components requires communication and synchronization mechanisms whose costs grow with system scale and failure scenarios [4]. In high-throughput environments processing millions of decision requests per day, even small coordination delays can accumulate into substantial latency and throughput degradation.Fragmentation also complicates incident response. When safety enforcement logic is distributed across multiple services and owned by different teams, organizations lack a unified mechanism for observing or controlling automated behavior during operational incidents. In such environments, mitigation often requires manual coordination across multiple teams operating with incomplete visibility into the system's overall state. Distributed systems theory has long established that maintaining consistent system behavior under partial failures becomes increasingly difficult as the number of independently operating components increases [4].Over time, these structural challenges erode organizational confidence in automation. As incidents accumulate and enforcement mechanisms remain inconsistent, organizations often respond by introducing manual approval gates or restricting automation scope. While such measures may reduce immediate operational risk, they also undermine the scalability and responsiveness that autonomous decision systems are intended to provide.The root cause of this fragmentation is not poor engineering practice but unclear ownership boundaries. When the responsibilities for decision generation, operational safety enforcement, and presentation logic are not explicitly separated, each product team is forced to implement its own operational safeguards. Without a shared enforcement layer governing execution across teams, autonomous decision systems inevitably fragment as they scale.

3. A Three-Layer Separation of Responsibility

A durable architecture for autonomous decision systems requires separating the responsibilities associated with **decision generation, execution safety, and presentation**. These responsibilities correspond to three distinct concerns: **decision intent, execution permission, and content selection**. While these concerns may appear closely related in small systems, they become increasingly difficult to manage when autonomous decisions operate across multiple teams, services, and customer-facing platforms.

In large-scale organizations, combining these responsibilities within the same service layer creates tight coupling between business logic, operational safeguards, and presentation behavior. Over time, this coupling increases maintenance complexity and limits the ability of teams to evolve their systems independently. Software architecture research has consistently shown that modular systems with well-defined responsibility boundaries experience lower defect propagation and reduced maintenance overhead compared with tightly coupled systems [5].

The first layer, **decision intent**, is owned by product teams responsible for specific domains or customer experiences. Product teams define which actions are appropriate for a given customer context by analyzing relevant signals, applying domain-specific logic, and specifying candidate actions. Decision intent therefore answers the fundamental question: *what action should be taken under a given set of conditions?* Because this logic evolves rapidly and depends on domain knowledge, placing decision intent within product teams preserves both iteration speed and accountability for outcomes.

The second layer governs **execution permission**, which determines whether a candidate action may be executed automatically under current system conditions. Execution permission addresses cross-cutting operational concerns such as blast radius limits, system health constraints, fallback behavior, and operational risk tolerance. These concerns apply across all product teams and therefore require consistent enforcement across the organization. Implementing these safeguards independently within each service introduces fragmentation and inconsistent operational behavior.

To address this challenge, we introduce an **autonomy platform** responsible for enforcing execution permission across all decision-producing services. Importantly, this platform does not evaluate business correctness or determine which actions should be taken. Instead, it determines whether a proposed action may be executed automatically under current operational conditions. Distributed systems research demonstrates that control structures responsible for coordination across multiple components often require centralized or logically centralized enforcement mechanisms in order to maintain consistent system behavior under concurrent workloads [6].

The third layer, **content selection**, operates within the presentation layer of the system. When a set of

candidate actions has been generated and approved for execution, the presentation layer determines which actions are shown to users, in what order, and under which contextual conditions. This layer is responsible for ranking, personalization, experimentation, and exposure management. Because presentation requirements vary significantly across channels and customer contexts, selection logic should remain independent of both decision intent and execution permission.

Separating these three responsibilities allows each layer to evolve independently while preserving system reliability and organizational scalability. Product teams retain control over domain-specific decision logic, the autonomy platform provides consistent operational enforcement, and presentation systems manage user-facing content without introducing dependencies on decision generation or safety governance. Software architecture principles emphasize that components with different reliability and availability requirements should be placed in separate failure domains to prevent cascading failures across the system [6].

When any two of these layers are combined, the failure modes discussed in the previous section begin to appear. Product services may embed operational safeguards directly within decision logic, presentation systems may introduce runtime dependencies on decision infrastructure, or safety enforcement mechanisms may become fragmented across teams. Maintaining clear separation between intent, permission, and selection therefore provides a structural foundation for scaling autonomous decision systems across organizations.

4. Control Plane and Runtime Enforcement

Effective enforcement of autonomous decision policies requires separating policy computation from policy application. This distinction is not merely an optimization technique but a structural requirement for systems operating under strict latency and throughput constraints. In autonomous decision environments, millions of execution requests may occur each day, and enforcement mechanisms must therefore operate within tight response time budgets.

The control plane is responsible for computing and distributing execution policies that govern autonomous behavior. This layer operates asynchronously and continuously aggregates system-level signals such as decision outcomes, operational budgets, system health indicators, and human override events. Based on these inputs, the control plane generates versioned execution policies that define the boundaries within which autonomous decisions may be executed. These policies are distributed and cached by runtime enforcement nodes before any request is processed.

The separation of control and execution logic is a well-established principle in distributed systems architecture. Software-defined networking (SDN) research demonstrates that decoupling policy control from runtime forwarding decisions enables centralized policy management while preserving high-performance execution at individual nodes [7]. By computing policies asynchronously and distributing them ahead of time, large-scale systems maintain consistent enforcement behavior without introducing latency on the request path.

At runtime, the enforcement layer performs only lightweight operations. Each decision request is evaluated against the locally cached execution policy using constant-time checks and local counters. No synchronous calls to external services are required during this process. Avoiding runtime coordination across services is critical in high-throughput systems, where even small synchronization delays can accumulate into significant latency and throughput degradation.

Research on scalable shared-memory systems similarly demonstrates that performance improves when coordination mechanisms minimize remote synchronization and rely on locally cached state whenever possible [8]. In multiprocessor systems, architectural optimizations such as caching and buffering reduce contention and allow parallel operations to proceed without serializing access to shared resources. Applying similar principles to autonomous decision enforcement allows policy checks to occur efficiently at runtime without introducing distributed coordination overhead.

This architecture ensures that enforcement overhead remains negligible even under high concurrency. Policies are computed asynchronously by the control plane, distributed to runtime systems, and applied locally during execution. As a result, enforcement behavior remains consistent across the system while preserving the low-latency performance required for large-scale customer-facing platforms.

5. Scheduled Execution as a Source of Latent Risk

Scheduled execution introduces a distinct category of risk in autonomous decision systems. Unlike real-time decisions, which are evaluated and executed immediately, scheduled decisions may be authorized hours or days before they are executed. During this interval, the operational environment in which the decision will execute may change significantly. System health conditions, operational

budgets, traffic patterns, or incident states may differ materially from those present at the time the decision was originally scheduled.

Treating scheduling as the moment of authorization therefore introduces a form of **latent risk**. A decision that was appropriate when scheduled may become unsafe or undesirable by the time it is executed. This challenge is particularly relevant in large-scale systems that support automated campaigns, batch operations, or long-running decision workflows.

Research in real-time systems demonstrates that the correctness of automated tasks depends on the **system state and resource conditions present at execution time rather than those present when the task was scheduled**. Classic real-time scheduling theory shows that tasks scheduled in advance must remain feasible under changing resource availability and system load conditions; otherwise, system correctness can no longer be guaranteed [9]. When there is a temporal gap between authorization and execution, the assumptions used to authorize a decision may no longer hold when the decision is executed.

Recent research on real-time scheduling for modern computing platforms similarly highlights the importance of validating task feasibility under dynamic system conditions, particularly in environments where workloads and resource availability evolve continuously [10]. Autonomous decision systems operating across scheduling horizons of hours or days are therefore structurally exposed to this risk unless execution permission is reevaluated at runtime. Scheduling should therefore be interpreted as a statement of **decision intent**, not as a guarantee of execution authorization.

Runtime enforcement mechanisms address this challenge by reevaluating execution permission immediately before the action is executed. Research on runtime verification and safety enforcement for autonomous systems similarly emphasizes the need for monitoring and validating system behavior during execution rather than relying solely on pre-execution validation [11]. Static scheduling analysis performed at submission time cannot anticipate system degradation, resource contention, or operational incidents that occur before execution. Autonomy platforms mitigate this risk by separating scheduling from execution permission. Scheduled actions are stored as candidate decisions but must pass runtime enforcement checks before execution. These checks typically involve evaluating the action against the current execution policy using lightweight constant-time policy lookups against locally cached policy state.

This design allows organizations to support long-horizon scheduling while preserving operational safety. Campaigns and automated workflows can be scheduled in advance, but execution remains governed by current system conditions rather than historical approvals. By evaluating execution permission at runtime, autonomous systems ensure that automated actions remain aligned with the present operational environment.

The temporal distance between decision authorization and execution influences the reliability of automated actions. As the scheduling horizon increases, the likelihood that environmental conditions have changed also increases. Table 4 illustrates this conceptual relationship between scheduling horizon length and latent risk exposure.

Illustrative Simulation Scenario

To illustrate the operational implications of the proposed framework, consider a simplified simulation scenario involving an autonomous decision system operating within a large-scale digital commerce platform. The system is responsible for scheduling automated promotional actions targeted at specific customer segments based on behavioral signals and historical engagement data.

Assume that the platform processes approximately five million customer interactions per hour across multiple services. Product teams can schedule automated promotional campaigns that execute at predetermined times. In this scenario, a product team schedules a promotional discount campaign targeting high-value subscribers. The campaign is authorized at 08:00 AM and scheduled to execute at 08:00 PM, creating a scheduling horizon of 12 hours between authorization and execution.

Under a conventional scheduling model, authorization occurs at the time the campaign is created. Once approved, the system executes the campaign automatically at the scheduled time without re-evaluating system conditions. During the 12-hour interval between scheduling and execution, however, several operational conditions may change. System monitoring may detect elevated error rates in downstream pricing services, traffic spikes may increase system load, or promotional budgets may become constrained due to concurrent campaigns.

In traditional architectures, scheduled campaigns execute regardless of these changes. This behavior exposes the system to latent risk, where decisions approved under earlier system conditions execute in a materially different environment. To illustrate this effect, a simulation was constructed to evaluate the relationship between scheduling horizon length and the probability that system conditions diverge from those assumed during scheduling. The simulation models a system processing 1,000 scheduled

decisions per hour, where the probability of environmental change increases as the scheduling horizon grows. Environmental change is defined as a deviation in system health metrics, budget constraints, or service availability that would invalidate the assumptions used during scheduling.The simulation demonstrates that as the scheduling horizon increases, the likelihood that environmental conditions diverge from those assumed during scheduling increases significantly. In the absence of runtime enforcement, these environmental changes translate directly into a higher probability of unsafe or suboptimal decision execution.Under the proposed autonomy framework, scheduled actions are treated as **expressions of intent rather than guaranteed authorization**. At execution time, the runtime enforcement layer evaluates each action against the current execution policy generated by the control plane. If system conditions violate policy constraints — for example due to degraded service health or exhausted operational budgets — the action is blocked or deferred.In the simulation, runtime enforcement reduces the probability of unsafe execution from **55% to approximately 4% for long-horizon scheduled decisions**, demonstrating how separating scheduling from execution permission can significantly reduce operational risk.

While the simulation is illustrative rather than predictive, it highlights the structural vulnerability introduced by long scheduling horizons and demonstrates the effectiveness of runtime enforcement in mitigating this risk.

Table 1: *Inter-Service Communication Complexity vs. Number of Microservices*

Approximate Number of Microservices	Communication Complexity Level	Maintenance Overhead Level
< 10	Low	Low
10-25	Moderate	Moderate
25-50	High	High
50-100	Very High	Substantial
>100	Critical	Very High

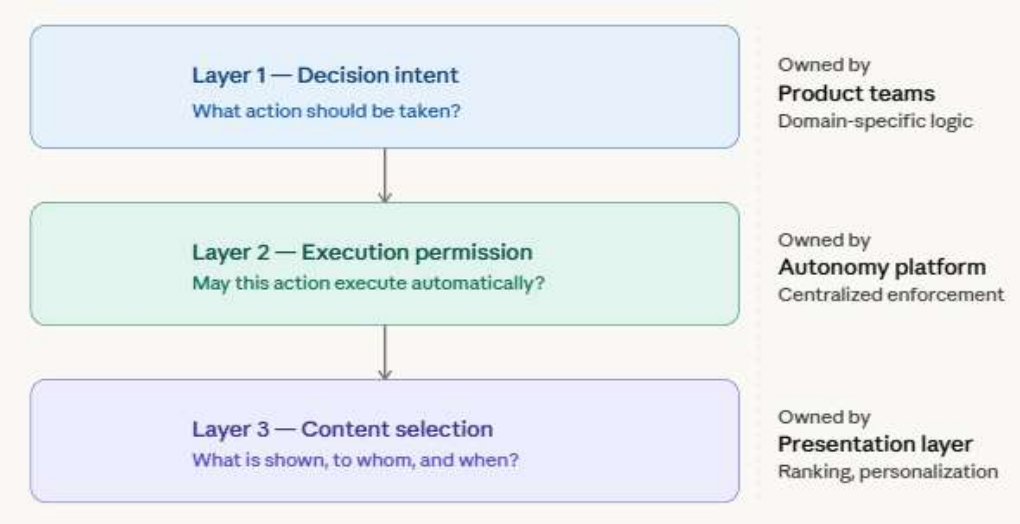

Table 2: *Impact of Separation of Concerns on Fault Propagation and Maintenance Effort [5, 6]*

Architecture Separation Level	Fault Propagation Scope	Maintenance Effort Level
Fully Coupled System	Failures propagate across most components	Very High
Partially Separated	Failures propagate across multiple subsystems	High
Moderately Separated	Failures contained within several components	Moderate

Largely Separated	Failures typically confined to individual modules	Low
Fully Separated (Intent–Permission–Selection)	Failures isolated within individual layers	Minimal

Table 3: *Relationship Between Scheduling Horizon and Latent Risk Exposure [9, 10]*

A	B	C	D
Scheduling Horizon	**State Freshness at Execution**	**Decision Outcome Reliability**	**Latent Risk Exposure**
Immediate	Fully Current	Very High	None
Sec - Minutes	Highly Current	High	Very low
Minutes–Hours	Mostly Current	Moderate	Moderate
Hours–Days	Partially Outdated	Reduced	High
Days or Longer	Outdated	Low	Very High

Table 4: *Simulation illustrating the relationship between scheduling horizon and latent risk exposure.*

Scheduling Horizon	Probability of Environmental Change	Risk of Unsafe Execution (No Runtime Enforcement)	Latent Risk Exposure
Immediate (0-1 min)	1%	1%	None
Short (1–30 min)	5%	5%	<1%
Medium (30 min - 4hrs)	18%	18%	~2%
Long (4-12hrs)	37%	37%	~3%
Very Long(12-24hrs)	55%	55%	~4%

6. Conclusions

As autonomous decision systems continue to expand across large-scale digital platforms, the challenges associated with coordinating automated decisions across distributed services, operational constraints, and evolving runtime environments become increasingly significant. While advances in machine learning and automation have enabled systems to generate decisions with high accuracy, the safe deployment of these decisions at organizational scale requires architectural mechanisms that address the complexity of real-world operational environments.

This work presented an architectural framework for scaling autonomous decision systems across organizational boundaries by separating decision responsibilities into three distinct layers: intent generation, execution permission, and action selection. By decoupling these responsibilities, the framework prevents the tight coupling between model outputs and execution logic that often leads to fragility in large distributed systems.

A central contribution of this work is the introduction of a control-plane–driven runtime enforcement model, which governs automated actions using continuously updated operational policies. Rather than treating scheduling or model inference as implicit authorization for execution, the proposed framework evaluates execution permission at runtime using current system conditions. This approach addresses several systemic risks associated with large-scale automation, including policy fragmentation, cross-team operational conflicts, and latent risks introduced by long scheduling horizons.

The framework also highlights the importance of treating scheduled actions as expressions of organizational intent rather than guaranteed execution commands. By reevaluating execution permission immediately prior to execution, autonomous systems can maintain alignment with evolving operational conditions such as service health, resource constraints, and budget limits. The illustrative simulation scenario demonstrates how runtime enforcement mechanisms can significantly reduce the probability of unsafe or misaligned automated actions.

Beyond individual system reliability, the proposed architecture provides a pathway for organizations to scale autonomous decision-making across multiple teams while maintaining centralized governance and operational safety. The separation between intent, permission, and execution creates a foundation for interoperable automation platforms that allow teams to contribute decision logic

without compromising system stability or policy compliance.

Future work may explore quantitative evaluation of runtime enforcement strategies under different workload patterns, as well as the integration of adaptive policy mechanisms that dynamically adjust operational constraints based on system behavior and historical outcomes. As organizations increasingly rely on automation to coordinate complex digital ecosystems, architectural approaches that combine autonomy with governance will be essential for ensuring both scalability and safety. In summary, this work contributes a scalable architectural model for governing autonomous decision systems, demonstrating how runtime policy enforcement and layered responsibility separation can enable organizations to safely deploy automation at enterprise scale.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] M. Herlihy and N. Shavit,*The Art of Multiprocessor Programming.*San Francisco, CA, USA: Morgan Kaufmann, 2008.Available: https://researchgate.net/publication/213876653_The_Art_of_Multiprocessor_Programmin

[2] D. Sculley et al.,"Hidden Technical Debt in Machine Learning Systems,"*Advances in Neural Information Processing Systems (NeurIPS)*, 2015.Available: https://papers.nips.cc/paper_files/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

[3] S. Newman, Building Microservices: Designing Fine-Grained Systems, O'Reilly Media, 2015.Link https://samnewman.io/books/building_microservices/

[4] D. Dolev,"The Byzantine Generals Strike Again," *Journal of Algorithms*, vol. 3, no. 1, pp. 14–30, 1982.Available: https://www.sciencedirect.com/science/article/abs/pii/0196677482900049

[5] D. L. Parnas,"On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.Available: https://dl.acm.org/doi/10.1145/361598.361623

[6] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges,"*IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[7] D. Kreutz et al.,"Software-Defined Networking: A Comprehensive Survey,"*Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.Available: https://ieeexplore.ieee.org/document/6994333

[8] K. Gharachorloo et al.,"Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors,"*Proceedings of the 17th Annual International Symposium on Computer rchitecture*, 1990.Available: https://ieeexplore.ieee.org/document/134503

[9] C. L. Liu and J. W. Layland,"Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment,"*Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973. Available: https://dl.acm.org/doi/10.1145/321738.321743

[10] A. Zou et al.,"A Survey of Real-Time Scheduling on Accelerator-Based Heterogeneous Architectures for Time-Critical Applications," *arXiv preprint*, 2025. Available: https://arxiv.org/abs/2505.11970

[11] R. Caldas et al.,"Runtime Verification and Field-Based Testing for ROS-Based Robotic Systems," *IEEE Transactions on Software Engineering*, 2024. Available: https://research.chalmers.se/publication/542744/file/542744_Fulltext.pdf