Article

A Hybrid Graph–Markov Model for Workload Generation in Load Testing

Dara Surya Varaprakash

Department of Computer Science, VIT-AP University, Amaravathi, India; varaprakash.23mis7328@vitapstudent.ac.in

Abstract

Load testing is a critical component of performance engineering, but traditional script-based methodologies often fail to accurately represent the dynamic, stochastic behavior of real users in modern distributed systems. As web applications grow in complexity, linear testing sequences leave critical execution paths untested, obscuring concurrency bottlenecks. This paper proposes a hybrid conceptual framework that integrates probabilistic navigation graphs with Markov transition models to simulate realistic, chaotic user behavior. The proposed model represents application workflows as directed graphs, employing Markov chains to dictate virtual user navigation across system states based on probabilistic weights. By shifting from deterministic scripting to stochastic workload generation, the framework theoretically increases state space coverage and path diversity while providing a more flexible representation of user navigation behavior. We detail the multi-layered system architecture, formalize the mathematical foundation of the traversal engine, and introduce rigorous analytical metrics including transition entropy and state coverage probability. Ultimately, this framework introduces a probabilistic graph traversal approach that enables the stochastic exploration of application state spaces and emergent concurrency behavior.

Keywords: load testing; performance engineering; Markov Chains; workload generation; graph theory; stochastic modeling

---

1. Introduction

Load testing is a critical component of performance engineering used to evaluate how software systems behave under varying levels of demand. Traditional load testing techniques typically rely on predefined scripts that simulate user actions through fixed sequences of requests. While these approaches are widely used, they often fail to accurately represent the dynamic and diverse behavior patterns exhibited by real users interacting with modern applications.

The rapid growth of cloud-native architectures, microservices ecosystems, and highly interactive web platforms has significantly increased the complexity of modern software systems. Unlike traditional monolithic applications, contemporary systems often involve dozens or even hundreds of loosely coupled services communicating through APIs and asynchronous messaging protocols. This architectural evolution introduces intricate performance dynamics where the behavior of individual services can propagate cascading effects throughout the system. As a result, accurately predicting system performance under load has become considerably more challenging.

Load testing has traditionally relied on deterministic scripting approaches where predefined sequences of requests are replayed against the system under test. Although these approaches are effective for validating specific workflows, they fail to capture the inherent variability present in real user behavior. In production environments, users do not follow perfectly linear navigation paths. Instead, they frequently deviate from expected workflows, revisit previously accessed pages, abandon sessions prematurely, or perform actions in unpredictable sequences. These stochastic interactions can generate traffic patterns that differ significantly from those produced by deterministic test scripts.

While Markov models have previously been used to analyze web navigation behavior, prior work has largely focused on user analytics and clickstream prediction. In contrast, this work applies stochastic navigation modeling specifically to the problem of performance testing. By framing application workflows as probabilistic traversal graphs executed by large populations of virtual users, the proposed framework transforms load testing from deterministic script replay into a probabilistic exploration of the application state space. This perspective enables the systematic generation of diverse navigation paths and emergent concurrency patterns that are difficult to reproduce with traditional script-based tools.

1.1. Problem Statement

Modern applications often involve multiple navigation paths, conditional workflows, and asynchronous service interactions. Script-based load testing approaches struggle to capture this variability. The maintenance overhead of updating static scripts every time a user interface changes is substantial. Furthermore, deterministic scripts generate highly predictable traffic, whereas real-world traffic is characterized by stochastic bursts, abandoned sessions, and chaotic navigation. Consequently, performance bottlenecks that occur under realistic user behavior—such as database deadlocks or thread pool exhaustion during highly concurrent, multi-path navigation—may remain undetected during pre-production testing.

1.2. Research Objectives and Contributions

Integrating graph structures with Markov transition probabilities provides a theoretical basis for the creation of dynamic workload models that simulate realistic navigation patterns. The primary contributions of this paper are fourfold. First, we propose a formal graph-based representation of application workflows tailored specifically for non-functional performance testing. Second, we introduce the integration of Markov transition matrices to simulate unpredictable, realistic user navigation behavior, including the mathematical modeling of session termination through absorbing states. Third, we establish a comprehensive four-layer conceptual architecture for dynamic workload generation. Finally, we introduce rigorous analytical metrics, such as path diversity and transition entropy, illustrating the framework's theoretical capacity to expose concurrency defects that are frequently missed by traditional load testing tools.

2. Related Work

Our proposed framework builds upon established research in performance engineering, system simulation, and stochastic modeling, synthesizing recent advancements in graph topologies and Markovian sequence analysis.

A. Limitations of Traditional Load Testing and Simulation Extensive research has highlighted the limitations of linear, script-based load testing frameworks in capturing the stochastic nature of modern web traffic. Traditional tools often rely on static workload models that fail to capture human-computer interaction accurately. While log-based workload simulation frameworks attempt to extract user behavior abstractions from session logs [2], and End-to-End (E2E) testing tools aim to automate workload generation [1], these approaches often require deployed systems to collect real user logs, making them unsuitable for early-phase architectural testing. Furthermore, automated test case generation from software requirements frequently utilizes model-based approaches, but these traditional models face challenges with state-space explosion and struggle to dynamically adapt to highly variable workloads [3].

B. Graph Topologies and Sequence Modeling To generate a realistic workload, the testing model must accurately reflect how human users navigate a system's underlying architecture. Directed Acyclic Graphs (DAGs) are extensively utilized across domains to model complex task dependencies and application topologies [4], and have been critical in profiling performance in decentralized, highly concurrent systems [5]. However, generating tests that accurately reflect chaotic user concurrency requires extending these static graphs with probabilistic dynamics. Furthermore, recent studies in decentralized

infrastructure demonstrate that translating complex ecosystems into graph representations allows for rigorous stress testing and the simulation of sudden peak transaction loads without requiring live deployment [11]. Analyzing clickstream sequences is a proven methodology for understanding user intent and forecasting complex navigational behavior in web applications [7]. Crucially, recent studies on learning the Markov order of paths in graphs demonstrate that user clickstreams and navigation trajectories are inherently constrained by the underlying application's topology and exhibit strong Markovian properties [6]. This provides a strong theoretical justification for overlaying Markov transition matrices onto structural application graphs to simulate realistic, multi-path user journeys.

C. Stochastic Frameworks in System Evaluation The application of stochastic processes is well-established for managing uncertainty in complex computing environments. For instance, Markov Decision Processes (MDP) are widely used to orchestrate decision-making and handle environmental unpredictability in autonomous robotics and systems [8]. In distributed edge and cloud computing, the integration of MDPs, probabilistic cellular automata, and predictive graph networks facilitates dynamic load balancing, real-time task offloading, and state prediction under fluctuating network conditions [9,10]. Our conceptual framework extends these stochastic principles from a system-management role into a proactive workload-generation role, utilizing Markov transitions to simulate the unpredictable stress required to expose concurrency bottlenecks.

D. Workload Modeling in Performance Engineering Workload modeling has long been recognized as a central challenge in performance testing and capacity planning. Early studies in web workload characterization demonstrated that user request patterns exhibit significant variability and temporal clustering, making deterministic workload generation insufficient for realistic testing environments. Research in web traffic modeling has frequently relied on stochastic processes to approximate user behavior, including Poisson arrival processes, Markov chains, and heavy-tailed distributions. Markov models in particular have been widely used to represent navigation sequences in web environments. By modeling each page or application state as a node and representing transitions as probabilistic edges, Markov chains enable the generation of realistic user sessions derived from historical interaction patterns. The framework proposed in this work bridges these perspectives by combining graph-based structural modeling with Markov-based behavioral modeling, providing a unified conceptual foundation for dynamic workload generation.

3. Hybrid Model Architecture

Figure 1 illustrates the conceptual architecture of the proposed hybrid workload generation framework. The architecture separates system structure, user behavior modeling, and workload execution into distinct layers.

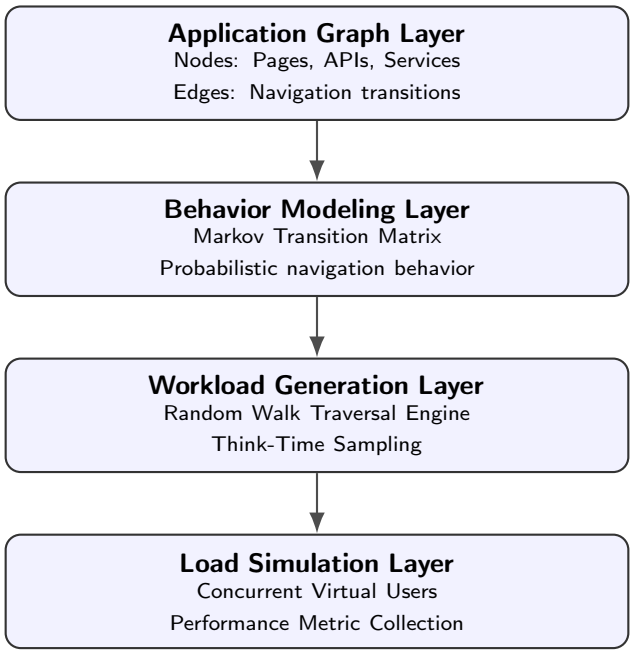

Figure 1. Hybrid Graph–Markov Load Testing Architecture.

The layered architecture proposed in this framework separates concerns between structural modeling, behavioral modeling, and workload execution. This separation improves conceptual clarity and enables each component to evolve independently. For instance, the application graph layer can be constructed using information extracted from system documentation, API specifications, or user interface navigation maps. Meanwhile, the behavior modeling layer can incorporate empirical data sources such as web analytics logs, user telemetry, or clickstream datasets.

The modular design also facilitates adaptability to different testing scenarios. In early development phases, transition probabilities may be assigned using heuristic estimates derived from expected user workflows. These probabilities can be seeded dynamically through heuristic seeding, where probabilities are assigned by performance engineers based on expected business flows, or through empirical seeding, where probabilities are derived from Google Analytics, server access logs, or historical clickstream data. As the system matures and real usage data becomes available, these probabilities can be refined using empirical observations. This flexibility allows the framework to remain applicable across multiple stages of the software development lifecycle.

Another advantage of the layered architecture lies in its scalability. Large enterprise systems often involve hundreds of potential interaction states. By separating graph representation from traversal logic, the workload generator can efficiently operate on large graphs while maintaining manageable computational complexity. Furthermore, because each virtual user independently performs stochastic traversal through the graph, the framework naturally supports high levels of concurrency. Finally, the architecture provides a foundation for integrating additional performance modeling techniques in future work, such as reinforcement learning methods to dynamically adjust transition probabilities based on observed system responses.

4. Formal Mathematical Model

The theoretical foundation of the workload generation engine relies on graph theory and stochastic processes.

4.1. Graph Representation and the Markov Property

The application under test is represented as a directed graph $G = (V, E, P)$, where V represents the set of nodes corresponding to application states, E represents the set of directed edges corresponding to transitions between states, and P represents the transition probability distribution. Each node $v_i \in V$

corresponds to a specific system interaction. User navigation follows the Markov property, where the probability of the next state depends only on the current state:

$$P(X_{n+1} = j \mid X_n = i)$$

4.2. Transition Probability Matrix and Absorbing States

The transition probabilities are represented as a matrix T. To accurately model real users, we must account for session abandonment. We define a set of *absorbing states* $A \subset V$ (e.g., Logout, Session Timeout, or Browser Close). Once a virtual user enters an absorbing state, the session terminates. For any absorbing state i, the transition probability is $p_{ii} = 1$ and $p_{ij} = 0$ for all $i \neq j$. For all non-absorbing (transient) states, the probabilities satisfy the condition that the sum of transition probabilities from any state i equals 1:

$$\sum_{j=1}^{n} p_{ij} = 1$$

4.3. Think-Time Modeling

Realistic load testing requires modeling the delay between requests. Let Δt be the think time between transitioning from node i to node j. We model Δt as a random variable following a log-normal distribution, which has frequently been used in prior workload and human-computer interaction studies to approximate human response times [2]. The function is defined as:

$$f(\Delta t) = \frac{1}{\Delta t \sigma \sqrt{2\pi}} \exp\left(-\frac{(\ln \Delta t - \mu)^2}{2\sigma^2}\right)$$

Here, parameters μ and σ vary depending on the complexity of node i, simulating the difference between reading a long article versus clicking a simple checkout button.

5. Analytical Properties of the Hybrid Model

To validate the framework as a rigorous foundation for performance engineering, we present several analytical properties that formally differentiate this model from deterministic testing approaches.

5.1. Path Diversity Analysis

A fundamental limitation of deterministic script testing is the lack of path diversity. For a given load testing script, the number of deterministic paths generated is exactly $P_{script} = 1$. By contrast, the proposed stochastic model organically scales the reachable execution space. For a graph $G = (V, E)$, the expected number of reachable paths of length k generated by the stochastic traversal is defined as:

$$P_{stochastic}(k) = \sum_{i=1}^{|V|} d_i^k$$

where d_i is the out-degree of node i. The number of reachable execution paths grows combinatorially with session length under stochastic traversal, whereas script-based testing generates only a fixed path per script.

5.2. State Coverage Probability

Load testing effectiveness can be analytically evaluated using state coverage probability. Given the transition matrix T, the probability distribution of user locations after n steps is given by $P^{(n)} = T^n$. Consequently, the expected visitation probability of node j after n transitions is formulated as:

$$C_j(n) = \sum_{i=1}^{|V|} \alpha_i P_{ij}^{(n)}$$

where α is the start-state distribution vector. A higher C_j indicates a higher theoretical test coverage for component j, providing the framework with a mathematically formal coverage metric.

5.3. Transition Entropy and Behavioral Variability

To formally quantify the behavioral variability—often colloquially described as user chaos—we introduce transition entropy. For a node i with outgoing transition probabilities p_{ij}, the transition entropy is defined as:

$$H(i) = -\sum_{j \in N(i)} p_{ij} \log p_{ij}$$

where $N(i)$ represents the set of neighbors reachable from node i. An entropy of $H(i) = 0$ denotes entirely deterministic behavior, equivalent to a rigid script. Higher entropy indicates greater navigation variability. To evaluate the global behavioral variability of the generated workload, the system-level entropy is defined as $H_{system} = \sum_{i \in V} \pi_i H(i)$, where π_i is the steady-state probability of node i. Transition entropy provides a formal measure of behavioral variability within the workload model, distinguishing deterministic script-based navigation from stochastic graph traversal.

5.4. Expected Load Distribution Across Application States

Beyond modeling individual user sessions, the hybrid graph–Markov framework enables analytical insights into long-term system behavior through steady-state analysis. In Markov chain theory, the steady-state probability vector $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ describes the long-run probability that a virtual user is located at node i. This steady-state distribution satisfies the equation $\pi T = \pi$ subject to the normalization condition $\sum_i \pi_i = 1$.

This distribution enables the estimation of expected request intensity across application components, providing a theoretical bridge between user navigation modeling and queueing theory. Given a total user request arrival rate λ, the expected request rate at node i is calculated as:

$$\lambda_i = \lambda \pi_i$$

Nodes with higher steady-state probability will receive greater workload intensity. This mathematical connection allows performance engineers to identify throughput bottlenecks and theoretically predict microservice load distribution under complex concurrent stress, similar to how edge betweenness centrality is utilized to pinpoint congestion bottlenecks in distributed blockchain transaction graphs [11].

5.5. Computational Complexity and Scalability

The proposed model maintains high computational efficiency, ensuring it is scalable for massive virtual user concurrency. The storage complexity of the application graph utilizing an adjacency list is $O(|V| + |E|)$, while storing the dense transition probability matrix requires $O(|V|^2)$ memory. During active workload generation, the traversal cost to compute the next probabilistic request is bounded by $O(d(v))$, where $d(v)$ is the out-degree of the current node. Finally, simulating an entire workload involving U concurrent virtual users over an average session length of k requires a time complexity of $O(U \cdot k)$. This linear scaling relative to user count theoretically assures that the framework can be practically implemented in industrial load generation engines.

6. Proposed Algorithmic Framework

The workload generation process can be expressed algorithmically as a stochastic traversal procedure executed independently by each virtual user. The algorithm relies on the transition probability matrix to determine the next state in the user session while incorporating probabilistic think-time delays between requests. This formulation demonstrates how the theoretical model described in the previous sections can be translated into a conceptual execution engine suitable for large-scale load testing environments.

Algorithm 1 Stochastic Workload Generation via Markov Traversal

Require: Graph $G = (V, E)$, Transition Matrix T, Start Node v_{start}
Ensure: Simulated Session Log with HTTP Requests
$current\_node \leftarrow v_{start}$
$session\_active \leftarrow$ **true**
while $session\_active$ **do**
    ExecuteHttpRequest($current\_node$)
    if $current\_node \in AbsorbingStates$ **then**
        $session\_active \leftarrow$ **false**
        break
    end if
    $\Delta t \leftarrow$ SampleLogNormalDistribution($\mu_{current}, \sigma_{current}$)
    Wait(Δt)
    $adjacent \leftarrow$ GetNeighbors($G, current\_node$)
    $probabilities \leftarrow T[current\_node]$
    $next\_node \leftarrow$ RandomWeightedSelect($adjacent, probabilities$)
    $current\_node \leftarrow next\_node$
end while

7. Theoretical Case Studies and Evaluation

Because the model relies on dynamic path generation rather than static definitions, it offers distinct theoretical advantages when analyzing system behavior under load.

7.1. Case Study: E-Commerce Concurrency Bottlenecks

Consider a traditional load test for an e-commerce site utilizing a linear script that follows the flow from Login to Search, View Product, Add to Cart, and finally Checkout. While this tests the optimal pathway, it fails to simulate users who frequently alternate between the Cart and Product pages to check shipping costs or modify items.

In our proposed model, illustrated in Figure 2, the edge between the Cart and Product nodes is assigned a defined transition probability, representing this alternating behavior. During execution with thousands of virtual users, the stochastic traversal organically generates complex, circular paths reflecting real-world uncertainty. Theoretically, this exposes database locking issues on the inventory table that only occur when a user repeatedly updates their cart state—a defect entirely missed by linear scripting.

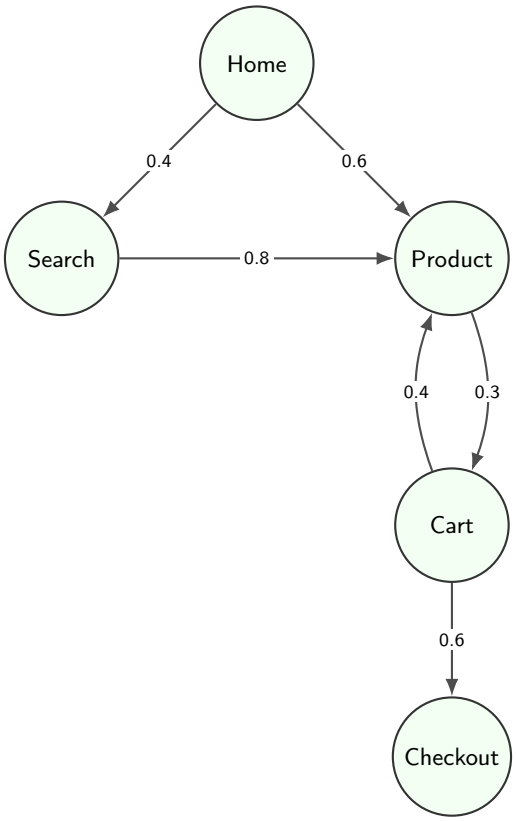

Figure 2. Example Application Navigation Graph with Markov Probabilities.

7.2. Evaluation Discussion

Beyond individual case studies, the hybrid graph–Markov model provides several theoretical advantages. One of the most significant benefits is the expansion of reachable execution paths during testing. In deterministic script-based approaches, each script corresponds to a single predefined navigation path. In contrast, the proposed stochastic traversal mechanism enables the generation of a combinatorially large number of potential user sessions. Because each transition decision is probabilistic, even a relatively small graph can produce thousands of unique navigation sequences, increasing the likelihood that rare or unexpected interaction patterns will occur during testing.

Another advantage relates to concurrency behavior. Performance defects in distributed systems often arise from the interaction of multiple concurrent requests rather than from individual request sequences. By generating diverse and overlapping request streams, the hybrid model increases the probability of exposing race conditions, resource contention, and database locking issues that might otherwise remain hidden. Although the current work focuses on conceptual modeling, these theoretical properties suggest that graph-based stochastic workload generation provides a much more comprehensive evaluation of system robustness compared to traditional deterministic testing strategies.

7.3. Evaluation Summary

Table 1. Theoretical comparison of load testing models.

Feature	Script-Based Testing	Hybrid Graph–Markov Model
Workflow Flexibility	Low (Rigid sequences)	High (Dynamic sequences)
User Behavior Realism	Moderate (Fixed think times)	High (Stochastic log-normal delays)
Path Diversity	Limited ($O(1)$ path per script)	Extensive (Probabilistic coverage)
Maintenance Overhead	High (Breaks on UI change)	Low (Update graph edges only)
Dynamic Workloads	Difficult	Natural

8. Conclusions & Future Work

This paper establishes a conceptual hybrid graph-based workload modeling framework integrating probabilistic navigation graphs with Markov transition models. By modeling application states as a directed graph and user behavior as a Markov chain with absorbing states and stochastic think-times, we provide a robust theoretical foundation for the next generation of performance testing tools. The formal graph-based representation allows for non-functional performance testing that closely mimics human interactions. The integration of Markov transition matrices combined with log-normal think-time delays facilitates realistic simulations of user navigation behavior. Furthermore, the four-layer architecture provides a scalable foundation for execution, enabling load testing platforms to theoretically expose concurrency defects and database deadlocks missed by traditional linear scripts. Future work will focus on the empirical implementation of this architecture, utilizing languages like Java or Python to build a multithreaded traversal engine and evaluate its efficacy against live, cloud-native microservices in production-like environments.

References

1. S. Di Meglio, L. L. L. Starace, and S. Di Martino, *Web App Performance Testing in Industrial Contexts: Supporting Workload Generation with E2E-Loader++*, 2025.
2. Y. Han, Q. Du, J. Xu, S. Zhao, Z. Chen, L. Cao, K. Yin, and D. Pei, *LWS: A framework for log-based workload simulation in session-based SUT*, The Journal of Systems & Software, Vol. 203, 111735, 2023.
3. Z. Yang, R. Huang, C. Cui, N. Niu, and D. Towey, *Requirements-Based Test Generation: A Comprehensive Survey*, ACM Transactions on Software Engineering and Methodology, 2025.
4. Y. Fang, H. Li, and W. Chang, *Directed Acyclic Graph Topology Generators: A Survey*, ACM Transactions on Embedded Computing Systems, 2025.
5. J. Shi, X.-Y. Bai, W.-Z. Zhang, P.-L. Li, K.-D. Wu, G.-L. Yang, and M.-T. Zhang, *Performance Modeling and Testing of DAG-Based Distributed Ledger Systems*, Journal of Computer Science and Technology, Vol. 40(6): 1593-1607, 2025.
6. L. V. Petrović and I. Scholtes, *Learning the Markov Order of Paths in Graphs*, Proceedings of the ACM Web Conference (WWW '22), 2022.
7. D. Y. C. Wang, L. A. Jordanger, and J. C.-W. Lin, *A Utility-Mining-Driven Active Learning Approach for Analyzing Clickstream Sequences*, 2024.
8. S. S. Goswami and S. Mondal, *Stochastic Models for Autonomous Systems and Robotics*, Spectrum of Operational Research, Vol. 3, Issue 1, pp. 215-237, 2026.
9. D. Sahu, Nidhi, R. Chaturvedi, S. Prakash, T. Yang, R. S. Rathore, L. Wang, S. Tahir, and S. T. Bakhsh, *Revolutionizing load harmony in edge computing networks with probabilistic cellular automata and Markov decision processes*, Scientific Reports, 15:3730, 2025.
10. K. Rajammal and M. Chinnadurai, *Dynamic load balancing in cloud computing using predictive graph networks and adaptive neural scheduling*, Scientific Reports, 15:22181, 2025.
11. G. Jayabalasamy, C. Pujol, and K. L. Bhaskaran, *Application of Graph Theory for Blockchain Technologies*, Mathematics, Vol. 12, No. 1133, 2024.