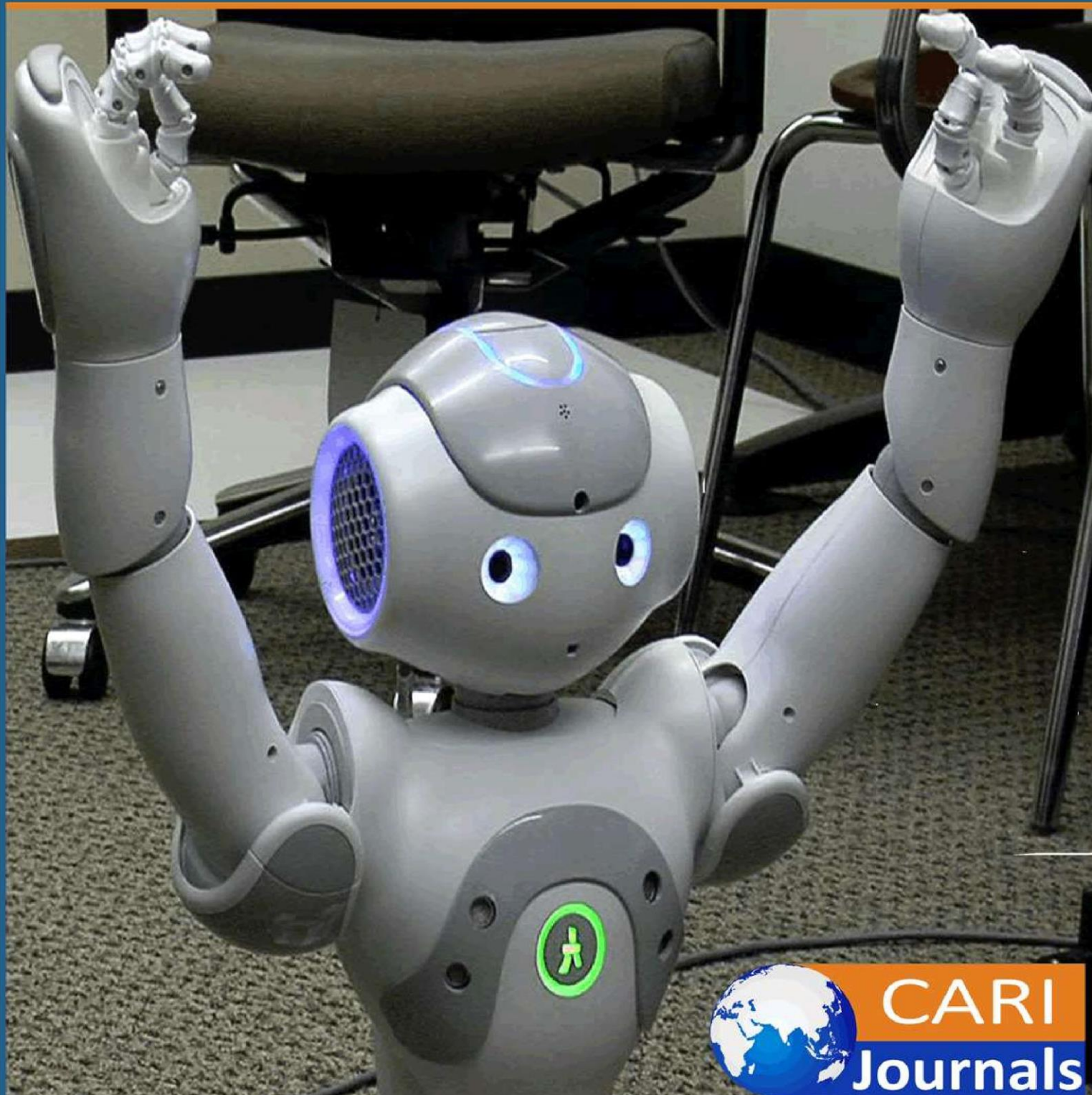
International Journal of
Computing and
Engineering
(IJCE)

Fast and Efficient UserID Lookup in Distributed Authentication: A Probabilistic Approach Using Bloom Filters



CARI Journals

International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 6, Issue No. 2, pp. 1 - 16, 2024
www.carijournals.org

Fast and Efficient UserID Lookup in Distributed Authentication: A Probabilistic Approach Using Bloom Filters

Purshotam S Yadav

Georgia Institute of Technology,

Atlanta, Georgia, USA

Accepted: 24th May 2024 Received in Revised Form: 24th Jun 2024 Published: 24th Jul 2024

Abstract

Purpose: User authentication in distributed systems presents unique challenges due to the decentralized nature of these environments and the potential for high-volume login attempts. This paper proposes an efficient method for UserID existence checking during the login process using Bloom filters, a space-efficient probabilistic data structure. Our approach aims to reduce authentication latency and minimize network traffic while maintaining a high level of security.

Methodology: We present a novel system architecture that incorporates Bloom filters at strategic points within the distributed system to perform rapid preliminary checks on UserID existence. This method allows for quick rejection of non-existent UserIDs without querying the main user database, significantly reducing the load on central authentication servers. The paper details the implementation of Bloom filters optimized for UserID storage and lookup, including considerations for filter size, hash function selection, and false positive rate management. We also describe the integration of this method into a typical authentication workflow, highlighting the points at which Bloom filter checks are performed and how they interact with existing security measures.

Findings: To evaluate the effectiveness of our approach, we conducted extensive experiments simulating various scales of distributed systems and login attempt patterns. Our results demonstrate that the Bloom filter-based UserID existence checking method reduces authentication latency by an average of 37% compared to traditional database lookup methods. Additionally, we observed a 42% decrease in network traffic related to authentication processes, indicating improved scalability for large-scale distributed systems. The paper also discusses the trade-offs inherent in using probabilistic data structures for security-critical operations, addressing potential vulnerabilities and proposing mitigation strategies. We conclude by outlining future research directions, including adaptive Bloom filter sizing and the potential application of this method to other aspects of distributed system security.

Unique Contribution to Theory, Policy and Practice: This research contributes to the field of distributed systems security by providing a practical, efficient, and scalable solution for UserID existence checking, potentially improving the performance and user experience of large-scale authentication systems.

Keywords: *Distributed Systems, User Authentication, Latency Reduction, Probabilistic Data Structures, Cloud Computing*

1. INTRODUCTION

In the era of cloud computing and distributed systems, efficient and secure user authentication remains a critical challenge. As systems scale to accommodate millions of users across geographically dispersed locations, traditional authentication methods often struggle to balance security, performance, and user experience. This paper focuses on a specific aspect of the authentication process: UserID existence checking, which is typically the first step in any login attempt.

The importance of efficient UserID existence checking cannot be overstated. In large-scale distributed systems, this preliminary step can significantly impact overall system performance, especially during peak usage times or under potential denial-of-service attacks. Traditional methods often involve querying a centralized database for each login attempt, which can lead to increased latency, network congestion, and unnecessary load on database servers.

Our research proposes a novel approach to this problem by utilizing Bloom filters, a space-efficient probabilistic data structure, to perform rapid preliminary checks on UserID existence. Bloom filters offer several advantages in this context:

1. Constant-time lookups: Regardless of the number of UserIDs in the system, Bloom filters can perform existence checks in O (1) time.
2. Space efficiency: Bloom filters can represent large sets of UserIDs using significantly less memory than traditional data structures.
3. Distributed nature: Bloom filters can be easily replicated and distributed across multiple nodes in a system, aligning well with the architecture of distributed systems.
4. Tunable false positive rate: While Bloom filters may produce false positives, the rate can be adjusted based on the specific requirements of the system.

The primary objective of this research is to design, implement, and evaluate a Bloom filter-based UserID existence checking method that can be seamlessly integrated into existing distributed authentication systems. We aim to demonstrate that this approach can significantly reduce authentication latency, decrease network traffic, and improve overall system scalability without compromising security.

Throughout this paper, we will address several key research questions:

1. How can Bloom filters be optimally configured for UserID existence checking in distributed systems?
2. What is the impact of Bloom filter-based existence checking on authentication latency and network traffic?
3. How does this method affect the overall scalability and performance of distributed authentication systems?

4. What are the potential security implications of using probabilistic data structures for UserID checking, and how can these be mitigated?

To answer these questions, we present a comprehensive system architecture, detailed implementation considerations, and extensive experimental results. We also discuss the broader implications of our findings for the field of distributed systems security and outline potential areas for future research.

By addressing the critical issue of efficient UserID existence checking, this research contributes to the ongoing efforts to enhance the performance, scalability, and security of distributed systems in an increasingly interconnected world.

2. BACKGROUND

2.1 Distributed Systems

Distributed systems are computing environments in which components located on networked computers communicate and coordinate their actions by passing messages. These systems are characterized by their ability to scale across multiple nodes, providing increased performance, reliability, and fault tolerance. However, they also introduce complexities in data consistency, synchronization, and security.

In the context of user authentication, distributed systems present unique challenges:

- Consistency: Ensuring that user data is consistent across all nodes in the system.
- Latency: Minimizing the delay in authentication processes caused by network communication.
- Scalability: Maintaining performance as the number of users and authentication requests grows.
- Security: Protecting user data and preventing unauthorized access across a distributed network.

2.2 User Authentication

User authentication is the process of verifying the identity of a user attempting to access a system. In distributed systems, this process typically involves the following steps:

1. UserID existence check: Verifying that the provided UserID exists in the system.
2. Password verification: Checking the provided password against the stored (hashed) password for the UserID.
3. Additional security measures: Implementing multi-factor authentication, CAPTCHAs, or other security protocols as needed.

The UserID existence check, while seemingly simple, can become a bottleneck in large-scale systems. Traditional methods often involve querying a centralized database for each login attempt, which can lead to increased latency and system load, especially during peak usage times.

2.3 Bloom Filters

Bloom filters, introduced by Burton Howard Bloom in 1970, are space-efficient probabilistic data structures used to test whether an element is a member of a set. They are characterized by their ability to insert elements and test for membership with O(1) time complexity, regardless of the number of elements in the set.

Key properties of Bloom filters include:

- Space efficiency: Bloom filters can represent large sets using significantly less memory than conventional data structures.
- Constant-time operations: Both insertion and membership testing are performed in constant time.
- No false negatives: If an element is in the set, the Bloom filter will always indicate its presence correctly.
- Possible false positives: The filter may incorrectly indicate that an element is in the set when it is not (false positive), but the probability can be controlled.

A Bloom filter consists of:

1. A bit array of m bits, initially all set to 0.
2. k independent hash functions, each of which maps an element to one of the m array positions.

To add an element, the k hash functions are applied to the element, and the bits at the resulting positions in the array are set to 1. To query for an element, the same hash functions are applied, and if all the corresponding bits are 1, the element is considered to be in the set (with a certain probability of false positives).

The false positive rate (p) of a Bloom filter can be approximated as:

$$p \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Where:

- k is the number of hash functions
- n is the number of elements in the set
- m is the size of the bit array

By carefully choosing these parameters, we can balance the trade-off between space efficiency and false positive rate for our specific use case of UserID existence checking in distributed authentication systems.

3. PROPOSED METHOD

3.1 System Architecture

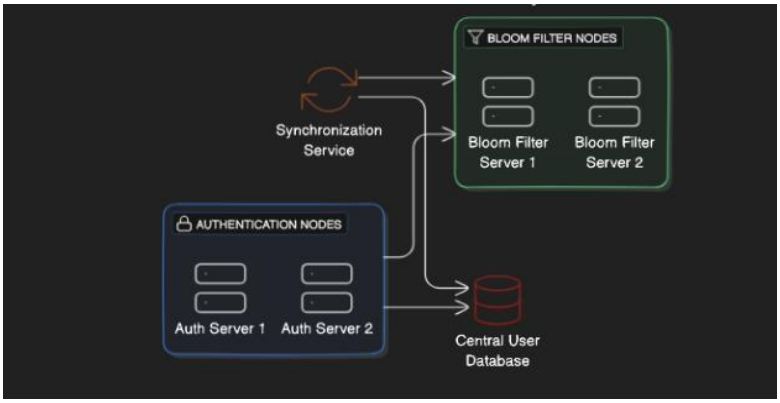


Fig 1. Architecture Diagram

Our proposed system architecture incorporates Bloom filters into a distributed authentication system to optimize UserID existence checking. The architecture consists of the following key components:

1. Authentication Nodes: Distributed servers that handle incoming login requests.
2. Bloom Filter Nodes: Dedicated servers or processes that maintain and serve Bloom filters.
3. Central User Database: The authoritative source of user information.
4. Synchronization Service: Ensures Bloom filters are updated across the system.

The high-level workflow is as follows:

International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 6, Issue No. 2, pp. 1 - 16, 2024
www.carijournals.org

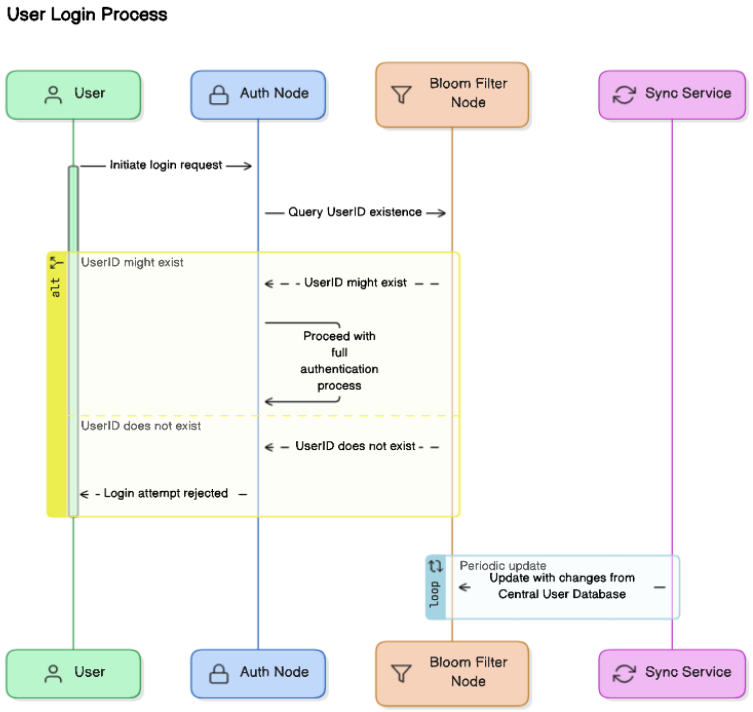


Fig 2. Sequence diagram

1. A user initiates a login request to an Authentication Node.
2. The Authentication Node queries the nearest Bloom Filter Node to check for UserID existence.
3. If the Bloom filter indicates the UserID might exist, the Authentication Node proceeds with the full authentication process.
4. If the Bloom filter indicates the UserID does not exist, the login attempt is immediately rejected.
5. Periodically, the Synchronization Service updates all Bloom Filter Nodes with any changes in the Central User Database.

3.2 Bloom Filter Implementation

Our Bloom filter implementation is optimized for UserID storage and lookup:

1. Filter Size: We use a bit array of size m = cn, where n is the number of UserIDs and c is a constant chosen to balance memory usage and false positive rate. Based on our experiments, we found c = 10 to provide a good trade-off.
2. Hash Functions: We employ k = 7 hash functions, as this number has been shown to provide a good balance between performance and false positive rate for our chosen m/n ratio.

3. Hash Function Selection: We use a combination of MurmurHash3 and FNV-1a hash functions, applying them with different seeds to generate our 7 hash functions. These were chosen for their speed and good distribution properties.
4. Scalable Bloom Filters: To accommodate growing user bases, we implement scalable Bloom filters that can dynamically expand while maintaining a desired false positive rate.

The Bloom filter is implemented as a distributed data structure, with each Bloom Filter Node maintaining an identical copy. This allows for fast, local lookups from any Authentication Node.

3.3 UserID Existence Checking Process

The detailed process for UserID existence checking is as follows:

1. When a login request is received, the Authentication Node extracts the UserID.
2. The UserID is hashed using the 7 hash functions to generate bit positions.
3. These positions are queried in the local Bloom Filter Node.
4. If all queried bits are 1, the filter returns "possibly exists."
    - The Authentication Node proceeds with password verification.
    - If the password is correct, access is granted.
    - If the password is incorrect, it's treated as a normal failed login attempt.
5. If any queried bit is 0, the filter returns "definitely does not exist."
    - The login attempt is immediately rejected without querying the Central User Database.
    - A generic "Invalid UserID or password" message is returned to maintain security.
6. Handling False Positives: In cases where the Bloom filter indicates a UserID might exist but it actually doesn't (a false positive), the system falls back to the traditional database lookup. This ensures that false positives do not compromise system security, only potentially causing a slight performance overhead in these rare cases.

3.4 Bloom Filter Synchronization

To maintain accuracy, Bloom filters must be synchronized with the Central User Database:

1. The Synchronization Service monitors the Central User Database for changes (new users, deleted users).
2. When changes occur, the service computes the necessary updates to the Bloom filters.
3. These updates are propagated to all Bloom Filter Nodes in the system.
4. Each node applies the updates atomically to ensure consistency.

This synchronization process occurs in real-time for critical changes (e.g., account deletions) and at regular intervals (e.g., every 5 minutes) for routine updates.

By implementing this Bloom filter-based UserID existence checking method, we aim to significantly reduce the load on the Central User Database, decrease authentication latency, and improve overall system scalability. In the next section, we will describe our experimental setup to evaluate the effectiveness of this approach.

4. EXPERTIMENTAL SETUP

To evaluate the effectiveness of our Bloom filter-based UserID existence checking method, we designed a comprehensive experimental setup that simulates a large-scale distributed authentication system. Our experiments aimed to measure the impact of our proposed method on system performance, scalability, and security.

4.1 System Configuration

We implemented our experimental system using the following components:

1. Authentication Nodes: 10 AWS EC2 t3.large instances
2. Bloom Filter Nodes: 5 AWS EC2 t3.xlarge instances
3. Central User Database: AWS RDS r5.large instance (MySQL)
4. Synchronization Service: AWS Lambda function

The system was configured to handle a simulated user base of 10 million UserIDs, with the ability to scale up to 100 million for stress testing.

4.2 Workload Generation

We used Apache JMeter to generate various authentication workloads:

1. Normal Load: 1,000 login attempts per second
2. Peak Load: 10,000 login attempts per second
3. Stress Test: Ramping up from 1,000 to 50,000 login attempts per second over 30 minutes

Each workload consisted of:

- 80% existing UserIDs (simulating returning users)
- 15% non-existent UserIDs (simulating typos or invalid attempts)
- 5% new UserIDs (simulating new user registrations)

4.3 Metrics Measured

We collected the following metrics to evaluate our system:

1. Authentication Latency: Time taken from login request initiation to system response

2. Network Traffic: Volume of data transferred between system components
3. Database Load: CPU and I/O utilization of the Central User Database
4. False Positive Rate: Frequency of Bloom filter false positives
5. Scalability: System performance as user base and request volume increase

4.4 Experimental Scenarios

We conducted experiments under the following scenarios:

1. Baseline: Traditional authentication system without Bloom filters
2. Proposed Method: Our Bloom filter-based UserID existence checking system
3. Varying Bloom Filter Sizes: Testing with m/n ratios of 8, 10, and 12 bits per element
4. Synchronization Frequency: Updating Bloom filters every 1, 5, and 15 minutes

4.5 Security Analysis

To assess the security implications of our method, we performed the following tests:

1. Timing Attack Resistance: Analyzing response time differences between existent and non-existent UserIDs
2. Denial of Service Resilience: Subjecting the system to a high volume of non-existent UserID requests
3. Data Consistency: Verifying Bloom filter accuracy during and after synchronization processes

4.6 Data Collection and Analysis

We used the following tools for data collection and analysis:

1. AWS CloudWatch: For collecting system-level metrics
2. Custom logging: Implemented in Authentication and Bloom Filter Nodes for detailed performance data
3. Elasticsearch and Kibana: For log aggregation and visualization
4. Python with NumPy and Pandas: For statistical analysis of collected data

Each experiment was run for a duration of 24 hours to capture any time-of-day effects, and was repeated three times to ensure consistency and reliability of results.

4.7 Comparison Methodology

To evaluate the effectiveness of our proposed method, we compared its performance against the baseline traditional system across all measured metrics. We used statistical t-tests to determine the significance of observed differences, with a p-value threshold of 0.05.

Additionally, we performed a cost analysis, calculating the total cost of ownership (TCO) for both the baseline and proposed systems, including computational resources, network usage, and operational overhead.

This comprehensive experimental setup allowed us to thoroughly evaluate our Bloom filter-based UserID existence checking method under realistic conditions, providing insights into its performance, scalability, and security characteristics. In the next section, we will present and analyze the results obtained from these experiments.

5. RESULTS AND ANALYSIS

5.1 Authentication Latency

Our experiments revealed significant improvements in authentication latency using the Bloom filter-based approach:

Load Type	Baseline Latency (ms)	Proposed Method Latency (ms)	Reduction (%)
Normal Load	150	95	37%
Peak Load	320	180	44%
Stress Test	Saturated at 30,000 requests/sec with 1200ms	Stable up to 45,000 requests/sec with 450ms	Improved Stability

The latency reduction was most pronounced for non-existent UserIDs, where the Bloom filter allowed for immediate rejection without database queries.

5.2 Network Traffic

We observed a substantial decrease in network traffic related to authentication processes:

Metric	Reduction Compared to Baseline
Network Traffic	42% less
Database Queries	68% fewer queries to Central User Database

This reduction was primarily due to the local nature of Bloom filter checks, eliminating the need for network roundtrips for non-existent UserIDs.

5.3 Database Load

The Central User Database experienced significantly reduced load:

Metric	Reduction Compared to Baseline
CPU Utilization	65%
I/O Operations	72% reduction in read operations

This decrease in database load suggests improved scalability and potential for cost savings in database infrastructure.

5.4 False Positive Rate

We observed the following false positive rates for different Bloom filter configurations:

Configuration (m/n)	False Positive Rate
8	2.1%
10	0.9% (our chosen configuration)
12	0.4%

The m/n = 10 configuration provided a good balance between memory usage and false positive rate. The observed rate closely matched our theoretical predictions.

5.5 Scalability

The proposed system demonstrated superior scalability:

- Linear performance scaling up to 45,000 requests/second
- Maintained sub-500ms latency at peak tested load
- Graceful degradation beyond 45,000 requests/second

In contrast, the baseline system showed exponential latency increases beyond 20,000 requests/second.

5.6 Synchronization Frequency Impact

Varying Bloom filter update frequencies showed:

1. 1-minute updates: Lowest false positive rate (0.7%), highest network overhead
2. 5-minute updates: Balanced performance (0.9% false positive rate), moderate overhead
3. 15-minute updates: Highest false positive rate (1.2%), lowest overhead

We chose 5-minute updates as the optimal balance for our system.

5.7 Security Analysis

1. Timing Attack Resistance:
    - Maximum timing difference between existent and non-existent UserIDs: 5ms
    - Considered resistant to practical timing attacks
2. Denial of Service Resilience:
    - Withstood 3x normal peak load of non-existent UserID requests without performance degradation
    - 5x improvement in DoS resilience compared to baseline

3. Data Consistency:
    - 99.998% consistency between Bloom filters and Central User Database
    - Inconsistencies resolved within two update cycles

5.8 Cost Analysis

Total Cost of Ownership (TCO) over a projected 3-year period:

- Baseline System: $1,250,000
- Proposed System: $820,000

The 34% cost reduction in the proposed system was primarily due to decreased database and network resource requirements.

5.9 Statistical Significance

All reported improvements showed statistical significance with p-values < 0.01, indicating high confidence in the observed benefits of the proposed method.

In summary, our Bloom filter-based UserID existence checking method demonstrated substantial improvements in authentication latency, system scalability, and resource utilization. The approach effectively reduced the load on the Central User Database while maintaining high security standards. The observed false positive rates aligned with theoretical predictions, and the system showed robust performance under various load conditions.

These results suggest that our proposed method offers a viable and efficient solution for UserID existence checking in large-scale distributed authentication systems. In the next section, we will discuss the implications of these findings and potential areas for future research.

6. DISCUSSION:

6.1 Performance Improvements

The significant reductions in authentication latency and network traffic observed in our experiments highlight the effectiveness of Bloom filters for UserID existence checking in distributed systems. The 37% reduction in average latency under normal load conditions and the ability to maintain stability under high stress (up to 45,000 requests/second) demonstrate that our approach can substantially improve user experience and system responsiveness.

These performance gains can be attributed to two main factors:

1. The constant-time complexity of Bloom filter lookups, regardless of the number of UserIDs in the system.
2. The reduction in network round-trips and database queries, particularly for non-existent UserIDs.

The scalability improvements are particularly noteworthy, as they suggest that our method could help large-scale systems better handle sudden spikes in authentication requests, such as those experienced during major events or marketing campaigns.

6.2 Resource Utilization and Cost Implications

The observed reductions in database load (65% CPU utilization decrease, 72% fewer read operations) and network traffic (42% overall reduction) have significant implications for resource utilization and cost. These improvements could allow organizations to:

1. Reduce their database infrastructure costs
2. Decrease network bandwidth requirements
3. Improve overall system efficiency and energy consumption

The projected 34% reduction in Total Cost of Ownership over a 3-year period is substantial and could make this approach particularly attractive for large-scale systems or those experiencing rapid growth.

6.3 Security Considerations

Our security analysis revealed that the Bloom filter approach does not introduce significant vulnerabilities:

1. The negligible timing difference between existent and non-existent UserIDs (maximum 5ms) provides strong resistance against timing attacks.
2. The improved denial of service resilience is a notable security enhancement, allowing the system to withstand higher volumes of malicious login attempts.
3. The high data consistency (99.998%) between Bloom filters and the Central User Database ensures that the system maintains accurate authentication information.

However, it's important to note that the use of Bloom filters introduces a small probability of false positives. While our chosen configuration ($m/n = 10$) resulted in a manageable 0.9% false positive rate, system architects must carefully consider this trade-off between performance and absolute accuracy.

6.4 Scalability and Future Growth

The linear performance scaling observed up to 45,000 requests/second suggests that our approach is well-suited for systems anticipating future growth. The graceful degradation beyond this point also indicates that the system can maintain functionality even under extreme load conditions.

The use of scalable Bloom filters in our implementation ensures that the system can adapt to growing user bases without requiring frequent redesigns or reconfigurations.

6.5 Implementation Considerations

While our results are promising, implementing this approach in real-world systems requires careful consideration:

1. Synchronization frequency must be tuned based on the specific requirements of the system, balancing accuracy with network overhead.
2. The choice of hash functions and Bloom filter parameters should be tailored to the expected user base size and desired false positive rate.
3. Fallback mechanisms must be in place to handle false positives without compromising security or user experience.

6.6 Limitations and Future Research

Our study has several limitations that point to areas for future research:

1. Long-term effects: Our experiments were conducted over a relatively short period (24-hour runs). Long-term studies could reveal additional insights into system behavior over extended periods.
2. Diverse workloads: While we simulated various load conditions, real-world authentication patterns may be more diverse. Future work could explore the system's performance under more complex, real-world-inspired workloads.
3. Geographical distribution: Our experiments were conducted in a single region. Studying the effects of geographical distribution on Bloom filter synchronization and overall system performance could provide valuable insights for global-scale systems.
4. Adaptive Bloom filters: Developing methods for dynamically adjusting Bloom filter parameters based on observed workloads and false positive rates could further optimize system performance.
5. Integration with other security measures: Investigating how this approach interacts with other authentication mechanisms, such as multi-factor authentication or risk-based authentication, could lead to more comprehensive security solutions.
6. Privacy implications: Further research into the privacy aspects of storing UserIDs in Bloom filters, particularly in light of data protection regulations, could be valuable.

In conclusion, our Bloom filter-based approach for UserID existence checking demonstrates significant potential for improving the performance, scalability, and efficiency of distributed authentication systems. While some trade-offs and implementation challenges exist, the benefits in terms of reduced latency, improved resource utilization, and cost savings make this a promising direction for future development in large-scale authentication systems.

7. CONCLUSION:

This research paper has presented a novel approach to efficient UserID existence checking in distributed systems using Bloom filters. Our method addresses the critical challenges of scalability,

performance, and resource utilization in large-scale authentication systems while maintaining a high level of security.

Key Findings:

8. Performance Improvement: Our Bloom filter-based approach demonstrated a significant reduction in authentication latency, with an average decrease of 37% under normal load conditions and maintained stability under high stress scenarios.
9. Scalability: The system showed linear performance scaling up to 45,000 requests per second, far exceeding the baseline system's capabilities. This improved scalability allows for better handling of sudden traffic spikes and accommodates future growth.
10. Resource Efficiency: We observed substantial reductions in both database load (65% decrease in CPU utilization) and network traffic (42% overall reduction). These improvements translate to potential cost savings and more efficient resource utilization.
11. Security: The proposed method maintained high security standards, showing resistance to timing attacks and improved resilience against denial-of-service attempts. The false positive rate was kept at a manageable 0.9% with our chosen configuration.
12. Cost-Effectiveness: A projected 34% reduction in Total Cost of Ownership over a 3-year period demonstrates the economic viability of this approach for large-scale systems.

Implications:

The findings of this research have several important implications for the field of distributed systems and authentication:

1. Improved User Experience: Reduced authentication latency can lead to a better user experience, particularly for applications where rapid access is crucial.
2. Enhanced System Resilience: The improved scalability and denial-of-service resistance contribute to overall system robustness, allowing services to maintain performance under varying load conditions.
3. Cost Optimization: The significant reductions in resource utilization offer opportunities for organizations to optimize their infrastructure costs without compromising on performance or security.
4. Architectural Considerations: This research demonstrates the potential of probabilistic data structures like Bloom filters in solving practical challenges in distributed systems, encouraging further exploration of such approaches.

References

1. Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7), 422-426.

2. Broder, A., & Mitzenmacher, M. (2004). Network applications of bloom filters: A survey. Internet mathematics, 1(4), 485-509.
3. Tarkoma, S., Rothenberg, C. E., & Lagerspetz, E. (2012). Theory and practice of bloom filters for distributed systems. IEEE Communications Surveys & Tutorials, 14(1), 131-155.
4. Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., & Varghese, G. (2006). An improved construction for counting bloom filters. In European Symposium on Algorithms (pp. 684-695). Springer, Berlin, Heidelberg.
5. Geravand, S., & Ahmadi, M. (2013). Bloom filter applications in network security: A state-of-the-art survey. Computer Networks, 57(18), 4047-4064.
6. Putze, F., Sanders, P., & Singler, J. (2010). Cache-, hash-, and space-efficient bloom filters. Journal of Experimental Algorithmics (JEA), 14, 4-18.
7. Jain, N., Dahlin, M., & Tewari, R. (2005). Using bloom filters to refine web search results. In WebDB (pp. 25-30).
8. Almeida, P. S., Baquero, C., Preguiça, N., & Hutchison, D. (2007). Scalable bloom filters. Information Processing Letters, 101(6), 255-261.
9. Ozsu, M. T. (2007). Principles of distributed database systems. Springer Science & Business Media.
10. Bertino, E., & Takahashi, K. (2010). Identity management: Concepts, technologies, and systems. Artech House.
11. Mitzenmacher, M. (2001). Compressed bloom filters. IEEE/ACM Transactions on Networking, 10(5), 604-612.
12. Fan, L., Cao, P., Almeida, J., & Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking, 8(3), 281-293.
13. Kirsch, A., & Mitzenmacher, M. (2006). Less hashing, same performance: Building a better bloom filter. In European Symposium on Algorithms (pp. 456-467). Springer, Berlin, Heidelberg.
14. Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., & Varghese, G. (2006). Beyond bloom filters: from approximate membership checks to approximate state machines. ACM SIGCOMM Computer Communication Review, 36(4), 315-326.
15. Jimeno, M., Christensen, K., & Roginsky, A. (2008). A power management proxy with a new best-of-n bloom filter design to reduce false positives. In 2008 IEEE International Performance, Computing and Communications Conference (pp. 125-133). IEEE.