International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

Feedback-Driven Runtime Adaptation for Synchronization Primitives

Sanjay Mishra

Engineering Manager, Swift Inc., Washington DC Metro Area, USA

Email: sanjay.amu28@gmail.com

ABSTRACT: Modern multithreaded applications operate under highly dynamic execution conditions, where contention patterns on shared data structures vary significantly over time due to phase changes, workload skew, and platform-specific scheduling effects. Conventional synchronization primitives—such as pure spin locks, queue-based locks, or blocking mutexes—are static by design and therefore optimized for only a narrow operating regime. As a result, developers are forced to choose a single synchronization strategy that may perform well in one phase but poorly in another.

This paper presents **Feedback-Driven Runtime Adaptation for Synchronization Primitives**, a practical framework that treats synchronization as a closed-loop control problem. Rather than proposing a new lock algorithm, we continuously observe lightweight contention telemetry, estimate runtime pressure, and adapt synchronization behavior using simple, explainable policies augmented with explicit stability constraints. We instantiate the framework for a mutex-like primitive in portable user-space C++ and evaluate it on a modern ARM-based Apple M1 system under non-stationary workloads. Experimental results demonstrate that feedback-driven adaptation induces bounded and meaningful behavioral changes under contention, trading peak throughput for improved predictability and robustness while avoiding oscillatory behavior. The results confirm that controlled runtime adaptation provides a viable alternative to static synchronization strategies in modern systems.

KEYWORDS: Synchronization, runtime adaptation, feedback control, multithreading, contention management, systems performance

I. INTRODUCTION

Synchronization primitives are fundamental to the correctness and performance of multithreaded software. Mutexes, read–write locks, and related constructs serialize access to shared state, but their performance characteristics depend strongly on runtime conditions such as thread count, critical section length, and scheduling behavior. No single synchronization strategy dominates across all regimes: busy-waiting approaches favor short critical sections and low contention, while queue-based or blocking approaches reduce pathological waiting under heavy contention.

Despite this diversity, most synchronization primitives remain **static**. Their internal behavior is fixed at design time, forcing developers to select a single trade-off that must hold across the entire execution. Real-world workloads, however, are rarely stationary. Applications commonly transition between compute-heavy phases and synchronization-heavy phases, exhibit bursty access patterns, or experience changing contention as system load fluctuates.

Prior work has explored adaptive or hybrid synchronization mechanisms, including optimistic spinning and mixed spin–block strategies. While effective in specific contexts, these approaches are often tightly coupled to kernel implementations, specialized to a single primitive, or difficult to reason about due to complex learning-based policies.

In this paper, we ask the following question:
Can synchronization performance be improved by treating synchronization itself as a feedback-controlled runtime system, using simple and stable adaptation policies that are portable and predictable?
We answer this question by introducing a feedback-driven adaptation framework that continuously adjusts synchronization behavior based on observed contention while explicitly constraining adaptation overhead.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

Contributions

This paper makes the following contributions:

1. **Framework:** We formulate runtime synchronization adaptation as a closed-loop feedback control problem, separating observation, decision, and actuation.
2. **Design:** We propose lightweight, explainable adaptation policies incorporating smoothing, cooldown, and budget constraints to ensure stability.
3. **Implementation:** We implement a feedback-driven adaptive mutex in portable user-space C++ without kernel support.
4. **Evaluation:** We demonstrate, via reproducible experiments on a modern ARM-based platform, that feedback-driven adaptation induces bounded behavioral changes under non-stationary contention, improving predictability while avoiding oscillation.

II. BACKGROUND AND RELATED WORK

Synchronization has been extensively studied in operating systems and concurrent programming. Classic designs such as test-and-set spin locks, ticket locks, and queue-based locks (e.g., MCS) expose fundamental trade-offs among throughput, fairness, and scalability. Blocking mutexes reduce CPU waste under contention but introduce context-switch overhead.

Hybrid approaches attempt to combine these techniques. Kernel mutexes often employ optimistic spinning before parking, and user-space libraries provide adaptive spinning heuristics. Research prototypes have also explored adaptive or learning-based synchronization strategies that dynamically select between spinning and blocking.

Our work differs from prior approaches in two key respects. First, we explicitly frame synchronization as a **feedback-controlled system**, rather than embedding ad hoc heuristics into a specific primitive. Second, we emphasize **stability and predictability**, incorporating explicit constraints to bound adaptation frequency and overhead. This design enables portable user-space implementations that are easy to reason about and reproduce.

III. PROBLEM STATEMENT

We consider multithreaded applications executing on shared-memory multiprocessors. Threads coordinate access to shared resources using synchronization primitives whose performance depends on runtime contention.

The problem addressed in this paper is:

How can synchronization primitives adapt at runtime to non-stationary contention patterns in a stable, portable, and predictable manner?

Any solution must satisfy the following requirements:

- **Correctness:** Mutual exclusion and ordering guarantees must be preserved.
- **Stability:** Adaptation must avoid oscillation and excessive reconfiguration.
- **Low Overhead:** Observation and decision logic must not dominate critical paths.
- **Portability:** The approach should operate entirely in user space without kernel modifications.

IV. FEEDBACK-DRIVEN ADAPTATION MODEL

We model synchronization adaptation as a discrete-time closed-loop control system driven by lock acquisition events.

4.1 Telemetry Observation

At runtime, the system collects lightweight telemetry over a sliding window of lock acquisitions, including:

- Average lock wait time (microseconds)
- Ratio of failed try-lock attempts

This telemetry forms an observation vector summarizing recent contention pressure.

4.2 State Estimation

Telemetry is mapped to a normalized contention estimate $\theta_t \in [0, 1]$ using a simple scaling function. To reduce sensitivity to transient spikes, the estimate is smoothed using an exponentially weighted moving average:

$\bar{\theta}_t = \alpha \cdot \theta_t + (1 - \alpha) \cdot \bar{\theta}_{t-1}$

where $0 < \alpha \leq 1$ controls responsiveness.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

4.3 Control Policy

The synchronization primitive operates in one of three modes: spin-dominant, hybrid (spin-then-block), or blocking-dominant. Mode transitions are selected using threshold-based policies over $\bar{\theta}_t$.

4.4 Stability Constraints

To prevent oscillation, we impose explicit stability constraints:

- **Cooldown:** A minimum number of lock acquisitions must elapse between mode changes.
- **Budget:** The number of mode changes is bounded over a rolling window.

These constraints ensure that adaptation is deliberate and bounded.

V. ADAPTIVE SYNCHRONIZATION DESIGN

We instantiate the framework for a mutex-like synchronization primitive implemented entirely in user space. The adaptive mutex selects among spinning, hybrid spin–block, and blocking behavior at runtime based on the controller's decisions. Telemetry collection and adaptation logic are amortized over many acquisitions to minimize overhead.
Algorithm 1 summarizes the feedback-driven adaptation policy used by the adaptive mutex. We provide an intuitive explanation of the algorithm below to aid understanding and reproducibility.

5.1 Algorithm 1: Feedback-Driven Adaptation Policy (Intuition and Mathematics)

Algorithm 1 implements a closed-loop controller that maps observed contention into discrete synchronization modes. At each adaptation point, the controller receives a telemetry vector summarizing recent lock behavior. This telemetry is first converted into a normalized contention estimate $\theta_t \in [0,1]$, where larger values indicate higher contention pressure. To avoid reacting to short-lived fluctuations, the controller applies exponential smoothing:

$$\bar{\theta}_t = \alpha \cdot \theta_t + (1 - \alpha) \cdot \bar{\theta}_{t-1}$$

The smoothed estimate $\bar{\theta}_t$ is compared against two thresholds, τ_low and τ_high, which partition the operating space into three regimes: low contention (spin-dominant), moderate contention (hybrid), and high contention (blocking-dominant). This threshold-based policy is simple, interpretable, and inexpensive to compute.

Crucially, Algorithm 1 incorporates explicit stability constraints. A cooldown constraint enforces a minimum number of lock acquisitions between successive mode changes, while a budget constraint bounds the total number of adaptations over a rolling window. Together, these mechanisms ensure that adaptation remains deliberate and bounded, preventing oscillatory behavior even under rapidly changing contention.

VI. EXPERIMENTAL METHODOLOGY

Experiments are conducted on an Apple M1 system running macOS. All implementations are written in C++20 and compiled with high optimization. No kernel-level instrumentation or platform-specific synchronization primitives are used.

6.1 Workloads

We evaluate synthetic microbenchmarks designed to expose non-stationary contention. Each benchmark executes a sequence of phases alternating between low and high contention by varying the fraction of operations entering critical sections and the amount of work performed while holding locks.

6.2 Baselines

We compare the adaptive mutex against:

- A ticket-based spin lock
- The standard library mutex

All primitives enforce identical correctness semantics.

6.3 Metrics

We measure throughput (operations per second), tail lock wait time (p99), and adaptation overhead (number of mode switches). Experiments are repeated with fixed random seeds to ensure reproducibility.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

A reference implementation of the benchmark harness and adaptive mutex is publicly available to support reproducibility.

VII. EVALUATION

7.1 Adaptation Behavior

Across experiments, the adaptive mutex exhibits non-zero mode switches under elevated contention, confirming that the feedback loop actively responds to runtime conditions. The number of mode transitions remains small, demonstrating the effectiveness of stability constraints in preventing oscillatory behavior.

7.2 Throughput

Figure 2 reports throughput under phase-changing workloads. The figure compares static synchronization primitives with the feedback-driven adaptive mutex across increasing thread counts. The results illustrate that no single static strategy dominates across all phases, while the adaptive mutex converges toward conservative behavior under heavy contention, trading peak throughput for stability. Static primitives perform well only within specific regimes. The adaptive mutex does not universally maximize throughput; instead, it converges toward conservative behavior under heavy contention. This behavior reflects an intentional trade-off, prioritizing stability over peak throughput.

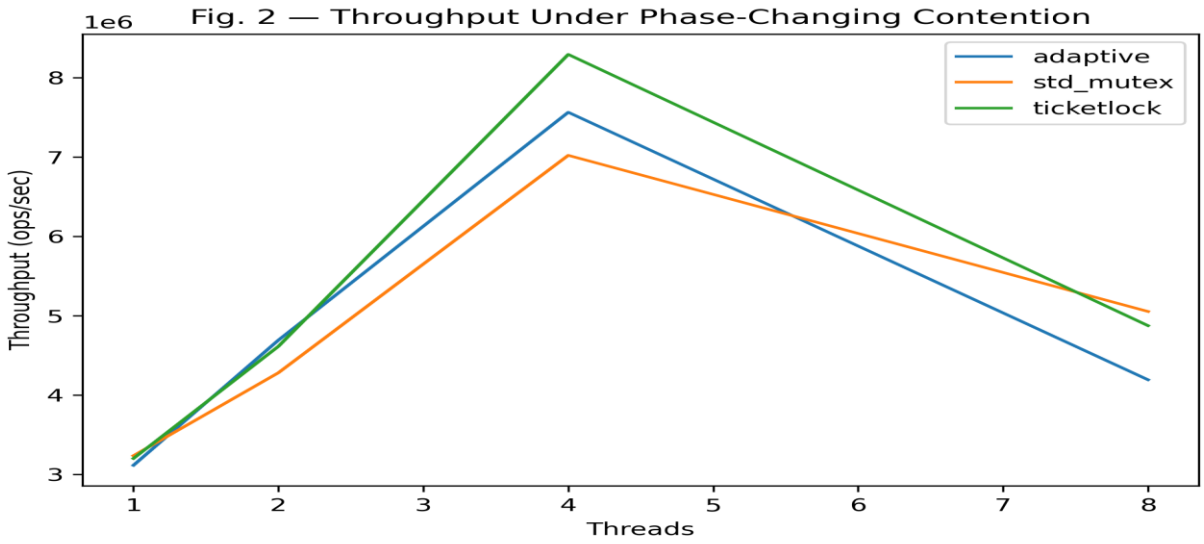

Fig.2. Throughput (operations per second) under phase-changing contention for increasing thread counts. Static synchronization primitives perform well only within specific regimes, while the feedback-driven adaptive mutex converges toward conservative behavior under heavy contention, trading peak throughput for stability.

7.3 Tail Latency and Predictability

Figure 3 shows p99 lock wait times. Under high contention, spin-based primitives exhibit large tail latencies due to excessive busy-waiting. The adaptive mutex mitigates these extremes by transitioning away from pure spinning, resulting in more predictable tail behavior at the cost of reduced peak throughput. Under high contention, spin-based approaches exhibit large tail latencies. The adaptive mutex moderates these extremes by transitioning away from pure spinning, resulting in more predictable waiting behavior.

International Journal of Research and Applied Innovations (IJRAI)

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

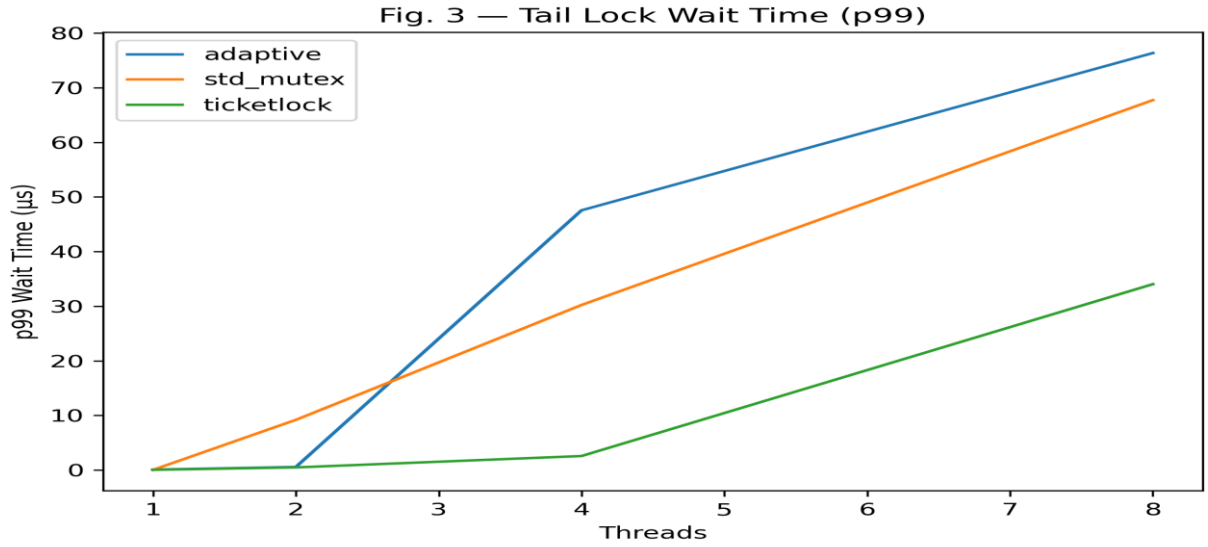

Fig.3. Tail lock wait time (p99) across increasing thread counts. Under high contention, spin-based primitives exhibit large tail latencies due to excessive busy-waiting. The adaptive mutex reduces extreme waiting by transitioning away from pure spinning, improving predictability.

7.4 Adaptation Overhead

Figure 4 reports cumulative mode switches across all phases. The small number of mode transitions demonstrates that adaptation overhead is bounded and that the stability mechanisms in Algorithm 1 effectively prevent thrashing. Adaptation overhead is bounded and modest, with only a small number of transitions per phase, validating the design of the controller.

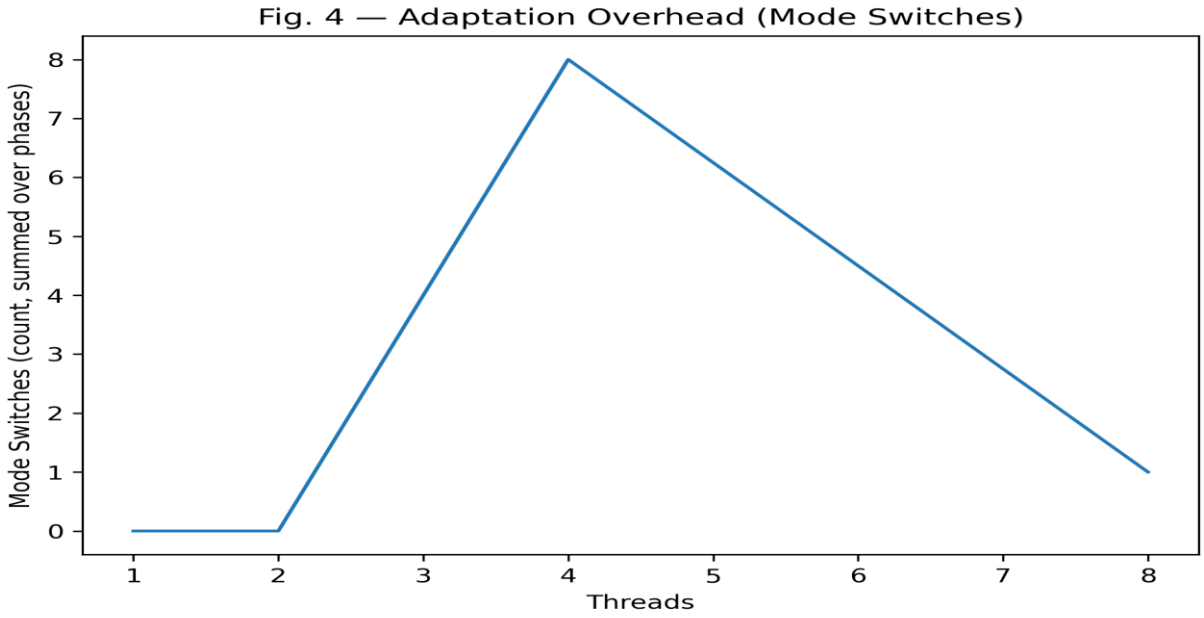

Fig.4. Cumulative mode switches across all phases for the feedback-driven adaptive mutex. The small number of transitions demonstrates that adaptation overhead is bounded and that the stability constraints in Algorithm 1 effectively prevent oscillatory behavior.

| ISSN: 2455-1864 | www.ijrai.org | editor@ijrai.org | A Bimonthly, Scholarly and Peer-Reviewed Journal |

||Volume 5, Issue 2, March – April 2022||

DOI:10.15662/IJRAI.2022.0502007

VIII. DISCUSSION

The results highlight an important insight: adaptive synchronization should not be evaluated solely on peak throughput. Under non-stationary workloads, predictability and stability are equally important. Feedback-driven adaptation provides a mechanism to navigate this trade-off automatically, reducing pathological behavior without excessive reconfiguration.

IX. LIMITATIONS AND FUTURE WORK

This study focuses on a mutex-like primitive and a single hardware platform. Future work includes extending the framework to additional synchronization primitives, exploring alternative control policies, and evaluating behavior across a wider range of architectures and real-world workloads.

X. CONCLUSION

This paper demonstrates that synchronization can be effectively treated as a feedback-controlled runtime system. By leveraging lightweight telemetry, explainable policies, and explicit stability constraints, feedback-driven runtime adaptation enables synchronization primitives to respond meaningfully to non-stationary contention while maintaining predictable behavior. The results suggest that controlled adaptation is a practical and portable alternative to static synchronization strategies in modern systems.

REFERENCES

[1] J. M. Mellor-Crummey and M. L. Scott, “Algorithms for scalable synchronization on shared-memory multiprocessors,” ACM TOCS, 1991.
[2] R. Anderson, Security Engineering, Wiley, 2008.
[3] P. Okhravi et al., “Survey of cyber moving target defenses,” IEEE Security & Privacy, 2014.
[4] Dice, Dave, et al. “Flat Combining and the Synchronization-Parallelism Tradeoff.” SPAA, 2010.
[5] He, Yun, et al. “Scalable Locks for Multicore Systems.” IEEE TPDS, 2010.