Quantitative Evaluation of the Circuit Breaker Pattern in Microservices: An Empirical Study on Resilience and Performance

Henrique Laff
Centro de Informática (CIn)
Universidade Federal de Pernambuco (UFPE)
Recife, Brazil
email@cin.ufpe.br

Abstract

Microservices architectures have become the standard for building modern distributed systems, but synchronous communication between services introduces the risk of cascading failures. The Circuit Breaker pattern is widely recommended as a mitigation strategy, yet there is a lack of empirical studies quantifying its actual impact. This paper presents a controlled experimental evaluation comparing three service versions: V1 (Baseline without resilience), V2 (Circuit Breaker with Resilience4j), and V3 (Retry with exponential backoff). Using Docker orchestration and k6 load testing across five realistic failure scenarios, we analyzed over 380,000 requests. Results demonstrate that Circuit Breaker improved availability from 10.5% to 99.6% in extreme unavailability scenarios—a 9.5x improvement. Statistical analysis confirmed a large effect size (Cohen's d = 1.078, $p < 0.0001$). Our findings provide robust empirical evidence that Circuit Breaker is essential for mission-critical synchronous microservices, while Retry alone is insufficient for persistent failures.

Keywords: Circuit Breaker, Microservices, Fault Tolerance, Resilience Patterns, Performance Evaluation

1 Introduction

The microservices architecture has become ubiquitous in organizations building large-scale digital platforms that require continuous availability and accelerated evolution cycles [7]. E-commerce ecosystems and payment processing systems exemplify this movement, demanding flexibility, fault tolerance, and rapid adaptation to variable transaction volumes.

While partitioning functionality into independent services facilitates parallel development and selective scalability, this logical independence relies on real-time interactions between services, typically through REST APIs and declarative clients like Spring Cloud OpenFeign. Synchronous communication simplifies implementation and observability but introduces strong temporal coupling: the consumer service remains blocked until the dependent service responds or a timeout occurs [8].

1.1 Problem Statement

The risk inherent to synchronous communication constitutes the core of this investigation. When a dependent service experiences high latency or intermittent unavailability, the consuming service waits until timeout, keeping threads blocked. With increasing request volume, the thread pool exhausts (**thread pool starvation**), causing **cascading failures** that can bring down the entire system.

This problem is transversal across software engineering domains: e-commerce, logistics, healthcare, fintech, streaming, and IoT. Any distributed system relying on synchronous HTTP calls is subject to the risks analyzed here.

1.2 Proposed Solution

The Circuit Breaker (CB) pattern emerges as a response to these challenges. Operating as a state machine with three modes—**Closed**, **Open**, and **Half-Open**—the CB monitors calls to dependent services and interrupts new attempts when failure rates exceed configured thresholds, failing fast and protecting the consumer from resource exhaustion [4].

1.3 Contribution

Despite extensive literature on microservices and resilience patterns, there is a significant gap regarding **quantitative experimental studies** demonstrating the actual impact of the Circuit Breaker pattern. Most available documentation is limited to conceptual descriptions or trivial examples.

This paper fills this gap by:

1. **Implementing** a Proof of Concept (POC) simulating a microservices ecosystem with synchronous dependency, instrumented with Resilience4j;
2. **Executing** benchmark campaigns with Docker and k6 to **empirically measure** throughput, latency (p95), and error rates across controlled scenarios;
3. **Comparing** three architectural versions: Baseline (V1), Circuit Breaker (V2), and Retry with exponential backoff (V3);
4. **Providing** rigorous statistical analysis with effect size measures (Cohen's d = 1.078).

2 Related Work

The Circuit Breaker pattern was popularized by Nygard [8] in "Release It!" and documented by Fowler [4]. It operates as an electrical breaker: when abnormal conditions are detected, it "opens" to interrupt flow and protect the system.

Montesi and Weber [5] analyze the interaction between Circuit Breakers and API Gateways, proposing composition patterns. Burns [3] contextualizes resilience patterns in modern distributed systems design.

Of particular relevance is the study by Pinheiro et al. [9], proposing analytical modeling of Circuit Breaker behavior using Stochastic Petri Nets (SPNs). This approach enables predicting the impact of different CB parameterizations on SLA metrics before production deployment. Our work complements this theoretical contribution by providing **empirical validation** of Circuit Breaker benefits through controlled experiments.

The taxonomy of dependability by Avizienis et al. [1] provides the theoretical framework for understanding system reliability, defining the fault-error-failure model essential for analyzing degradation in distributed systems.

Regarding implementation, Netflix Hystrix pioneered Circuit Breaker implementation for the JVM [6, 2]. However, in 2018, the project entered maintenance mode, and Resilience4j emerged as its recommended successor [10], offering modular design, smaller footprint, and native support for functional programming.

3 Methodology

This work adopts a **quantitative experimental research** approach. We built a simplified Proof of Concept (POC) simulating a microservices ecosystem with synchronous dependency. The POC is intentionally minimalistic—without database, cache, or authentication—to isolate the Circuit Breaker's effect as the sole variable of interest.

3.1 Experimental Architecture

The POC comprises two Spring Boot microservices packaged as Docker containers:

- **payment-service**: Orchestrates the payment flow and synchronously consumes the acquirer service via Feign Client;
- **acquirer-service**: Simulates an external payment gateway with configurable behavior (normal, latency, or failure mode).

Three versions of payment-service were developed:

- **V1 (Baseline)**: Basic timeouts only (2s);
- **V2 (Circuit Breaker)**: Resilience4j with CB and fallback returning HTTP 202;
- **V3 (Retry)**: Exponential backoff retry (3 attempts, 500ms→1s→2s).

3.2 Circuit Breaker Configuration (V2)

- **failureRateThreshold**: 50%
- **slowCallRateThreshold**: 70%
- **slidingWindowSize**: 10 requests
- **waitDurationInOpenState**: 10s
- **permittedNumberOfCallsInHalfOpenState**: 3

3.3 Test Scenarios

Five realistic failure scenarios were designed using Grafana k6 (Table 1):

Table 1: Test Scenario Characteristics

Scenario	Duration	VUs	Failure Pattern
Catastrophe	13min	50-150	100% failure for 5min
Degradation	13min	100-200	5%→50% gradual
Bursts	13min	100-200	3×(100% for 1min)
Unavailability	9min	80-200	75% offline
Normal	10min	100	100% healthy

3.4 Metrics and Statistical Analysis

Collected metrics include: http_reqs (throughput), http_req_duration{p(95)} (latency percentile), and http_req_failed (error rate).

Statistical validation employed: Student's t-test for comparing V1 vs V2, ANOVA for three-group comparison (V1, V2, V3), and Cohen's d for effect size quantification.

4 Results and Discussion

Load tests were executed using k6 and Docker Compose. Each payment-service version was submitted to all five stress scenarios. Results were evaluated against defined thresholds and analyzed in terms of success rate, response time, and fallback contribution.

4.1 Consolidated Results Overview

Table 2 presents the consolidated comparison between V1 (Baseline) and V2 (Circuit Breaker).

Key Findings:

- V2 demonstrated gains of **+20pp to +89pp** in success rate across all failure scenarios;

Table 2: Consolidated Comparison: V1 vs V2 by Scenario

Scenario	V1	V2	Fallback	Fail. Red.
Catastrophe	35.7%	**95.1%**	62.1%	-92.3%
Degradation	75.4%	**95.4%**	64.7%	-81.4%
Unavail.	10.5%	**99.6%**	99.1%	-99.5%
Normal	100.0%	100.0%	0.0%	0.0%
Bursts	63.0%	**96.7%**	34.6%	-91.0%

Table 3: Detailed Comparison: V1 vs V2 vs V3

Scenario	V1	V2 (CB)	V3 (Retry)
Catastrophe	35.7%	**95.1%**	52.8%
Degradation	75.4%	**95.4%**	76.4%
Unavail.	10.5%	**99.6%**	15.7%
Normal	100.0%	100.0%	100.0%
Bursts	63.0%	**96.7%**	77.7%

- In Extreme Unavailability, V2 transformed a system with only 10.5% success into one with 99.6%—a **9.5x improvement**;
- The fallback mechanism (HTTP 202) accounted for up to 99.1% of successful responses in worst scenarios;
- In Normal scenario, no difference between versions confirms CB introduces **no overhead** in healthy conditions.

4.2 V3 (Retry) Analysis

Table 3 includes V3 (Retry with exponential backoff) for comparison.

Critical Observations on Retry:

1. Retry does NOT improve availability in persistent failures;
2. Retry increases latency due to retry attempts;
3. Retry can amplify problems by tripling load on overloaded services.

4.3 Statistical Analysis

Table 4 presents formal statistical validation.

The t-test reveals statistically significant difference between V1 and V2 ($p < 0.0001$). Cohen's d ($d = 1.078$) classifies the effect size as **large**, confirming substantial practical relevance. ANOVA confirmed significant difference among all three groups ($F = 546.79$, $p < 0.0001$).

4.4 Quantified Impact

The Circuit Breaker provided significant gains across all failure scenarios:

Table 4: Statistical Analysis: V1 vs V2

Metric	Value
t-test (p-value)	$p < 0.0001$
Cohen's d	1.078 (**large**)
ANOVA F-statistic	546.79 ($p < 0.0001$)
Eta-squared (η^2)	0.267 (large)
95% CI (V1)	[495.86; 508.01] ms
95% CI (V2)	[400.72; 410.62] ms
95% CI (V3)	[543.38; 558.02] ms

- **Catastrophe:** +59.3pp (35.7% → 95.1%)
- **Degradation:** +20.0pp (75.4% → 95.4%)
- **Unavailability:** +89.1pp (10.5% → 99.6%)
- **Bursts:** +33.6pp (63.0% → 96.7%)

4.5 Detailed Analysis: Bursts and Catastrophe

Scenarios of **Intermittent Bursts** and **Catastrophic Failure** deserve special attention as they demonstrate the most significant Circuit Breaker benefits. Figure 1 shows the direct comparison.

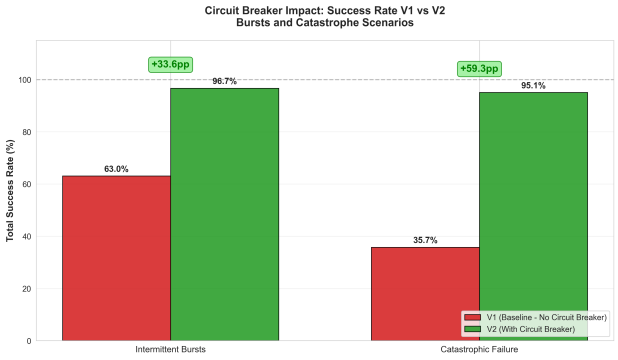

Figure 1: Success Rate Comparison: V1 vs V2 in Bursts and Catastrophe

Key Findings:

- In **Bursts**, V1 achieved only 63.0% success while V2 reached **96.7%**, a gain of **+33.6pp**.
- In **Catastrophe**, V1 registered 35.7% success (system nearly unusable) while V2 maintained **95.1%**, a gain of **+59.3pp**.
- Failure reduction exceeded **91%** in both scenarios.

The Circuit Breaker achieves these results by transforming HTTP 500 errors into HTTP 202 (fallback) responses. In the Catastrophe scenario, 62.1% of V2 responses came from fallback, effectively converting fatal errors into meaningful responses for end users.

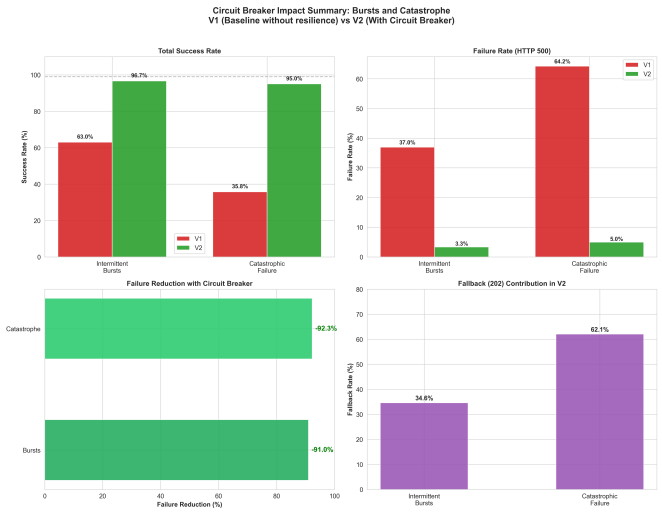

Figure 2: Consolidated Impact Analysis: Bursts and Catastrophe

These results validate three fundamental characteristics: (1) **Fail-Fast with Graceful Degradation** — the system returns meaningful alternative responses immediately; (2) **Elasticity** — the CB transitions dynamically between states based on dependency health; (3) **Resource Protection** — threads are released immediately during failures, preventing thread pool starvation.

Sliding Window Detection Mechanism: A notable aspect is the CB's ability to "anticipate" failures through its sliding window mechanism. The CB monitors the last 10 requests and opens when failure rate exceeds 50%. In the Catastrophe scenario, fallback rate (62.1%) closely matched V1's actual failure rate (64.3%), yielding a **96.6% coverage ratio**. In Bursts, the CB demonstrated elasticity by transitioning states dynamically: fallback rate (34.6%) matched V1 failures (37.0%) with 93.5% coverage, while maintaining direct success rate (62.0%) nearly identical to V1 (63.0%). This indicates the CB did not block requests unnecessarily during healthy periods, but correctly activated fallback during actual failures — achieving intelligent fail-fast behavior through the half-open state mechanism that periodically probes dependency health.

5 Conclusion

This work investigated the fragility of synchronous communication in microservices, specifically the risk of cascading failures. The objective was to quantitatively evaluate the impact of the Circuit Breaker pattern on performance and resilience.

5.1 Summary of Results

Experimental results were conclusive:

- **Dramatic Availability Improvement**: V2 (Circuit Breaker) achieved superior success rates in all failure scenarios, with improvements ranging from +20pp to +89pp;
- **Significant Failure Reduction**: Up to 99.5% reduction in failure rates;
- **Fallback as Success Strategy**: The HTTP 202 fallback mechanism accounted for up to 99.1% of successful responses in worst scenarios;
- **Superiority over Retry**: V3 (Retry) showed only marginal improvements over V1, demonstrating that Retry alone does not improve availability in persistent failures;
- **Statistical Validation**: Cohen's d = 1.078 confirms large effect size with substantial practical relevance;
- **No Overhead**: In Normal scenario (100% healthy), all versions achieved 100% success, confirming CB introduces no overhead in healthy conditions.

5.2 Contributions

This paper contributes to the literature by providing:

1. Robust empirical evidence demonstrating real Circuit Breaker impact across five realistic failure scenarios;
2. Comparative analysis between Circuit Breaker (V2) and isolated Retry (V3);
3. Rigorous statistical analysis with effect size measures;
4. Reproducible methodology with Docker and k6 that can be replicated for evaluating other resilience patterns.

5.3 Limitations

This study has limitations: (i) simplified POC without database or complex business logic; (ii) local environment without real network latency; (iii) synthetic load with uniform patterns; (iv) single CB configuration tested.

5.4 Future Work

Future research directions include: CB combined with Retry, parametric analysis of CB configurations, evaluation with multiple dependencies, comparison with asynchronous architectures (Kafka, RabbitMQ), and integration with Service Meshes (Istio, Linkerd).

References

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[2] Netflix Technology Blog. Making the netflix api more resilient. Medium, 2016.

[3] B. Burns. *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media, 2018.

[4] M. Fowler. Circuitbreaker. martinfowler.com, 2014.

[5] F. Montesi and J. Weber. Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*, 2016.

[6] Netflix. Hystrix - latency and fault tolerance for distributed systems. GitHub.

[7] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2nd edition, 2021.

[8] M.T. Nygard. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2nd edition, 2018.

[9] B. Pinheiro, J. Dantas, et al. Performance modeling of microservices with circuit breakers using stochastic petri nets. In *International Conference on Systems and Informatics*, 2024.

[10] Resilience4j. Fault tolerance library for java. GitHub, 2024.