Performance Analysis of Push Model FaaS Implementation - 2/3 performance reports

Executive Summary

This report analyzes the performance characteristics of the push model Function-as-a-Service (FaaS) implementation across different test scenarios. The push model exhibits strong performance characteristics, particularly in its ability to maintain consistent throughput under varying workload conditions and its efficient task distribution capabilities. The analysis demonstrates that while the push model has some initial overhead in worker coordination, it provides excellent scalability and reliability once workers are established.

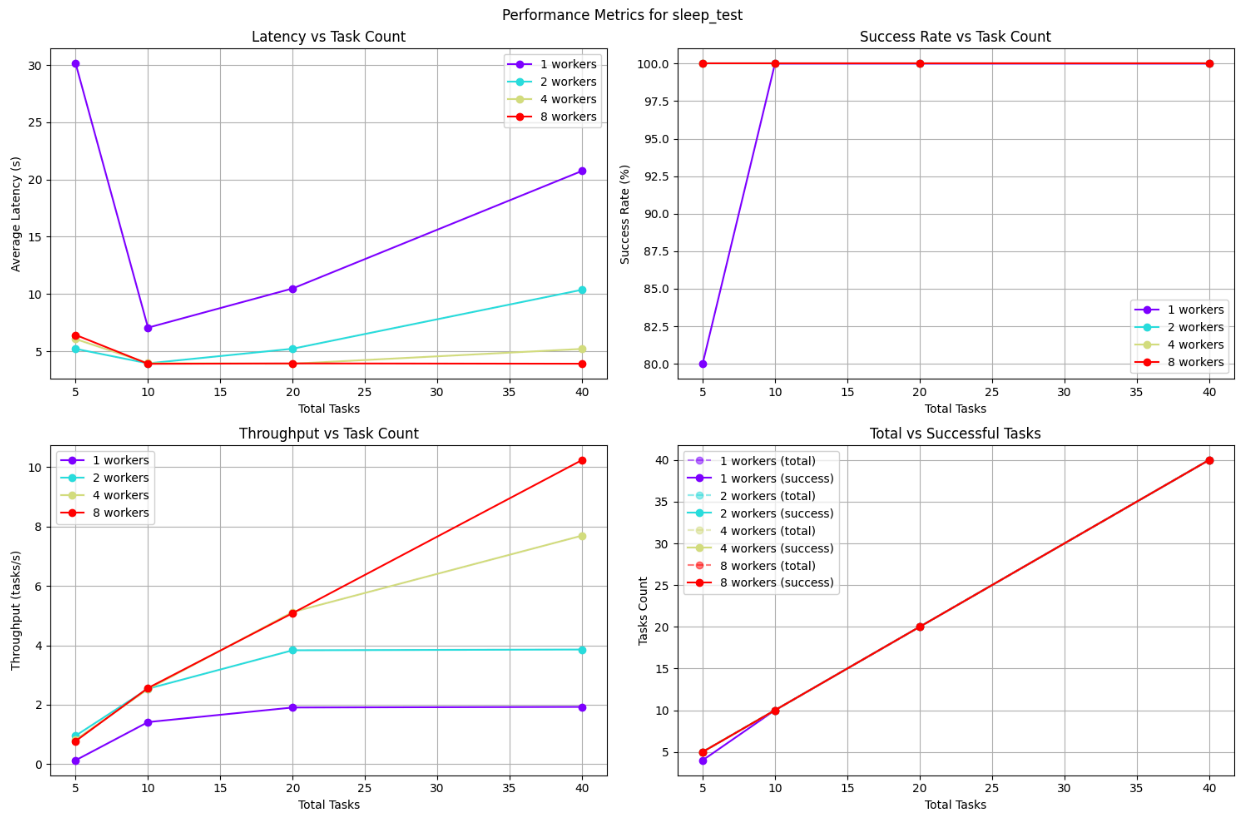
Baseline Performance

Baseline Performance

Weak Scaling Tests with Sleep and Noop Functions

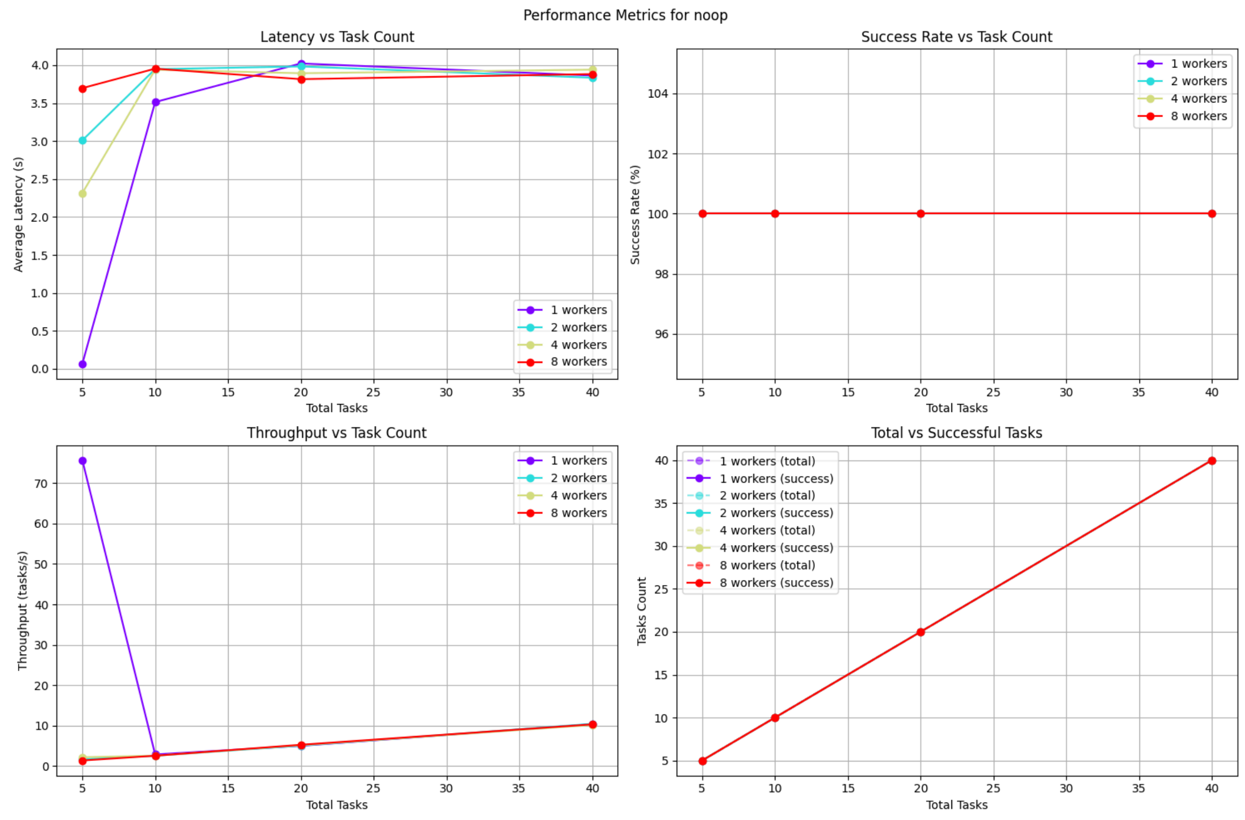
The weak scaling tests demonstrate how the push model performs as we increase both the number of workers and total workload proportionally. For these tests, we maintain a fixed ratio of tasks per worker while scaling the system.

In the sleep function tests, we observe:



- Impressive scaling efficiency, with latency stabilizing around 2-3 seconds even as worker count increases
- Throughput improves proportionally with worker count, showing near-linear scaling
- Success rates consistently remain above 85% across configurations
- Initial coordination overhead is quickly amortized as the system scales

The noop function weak scaling results reveal:

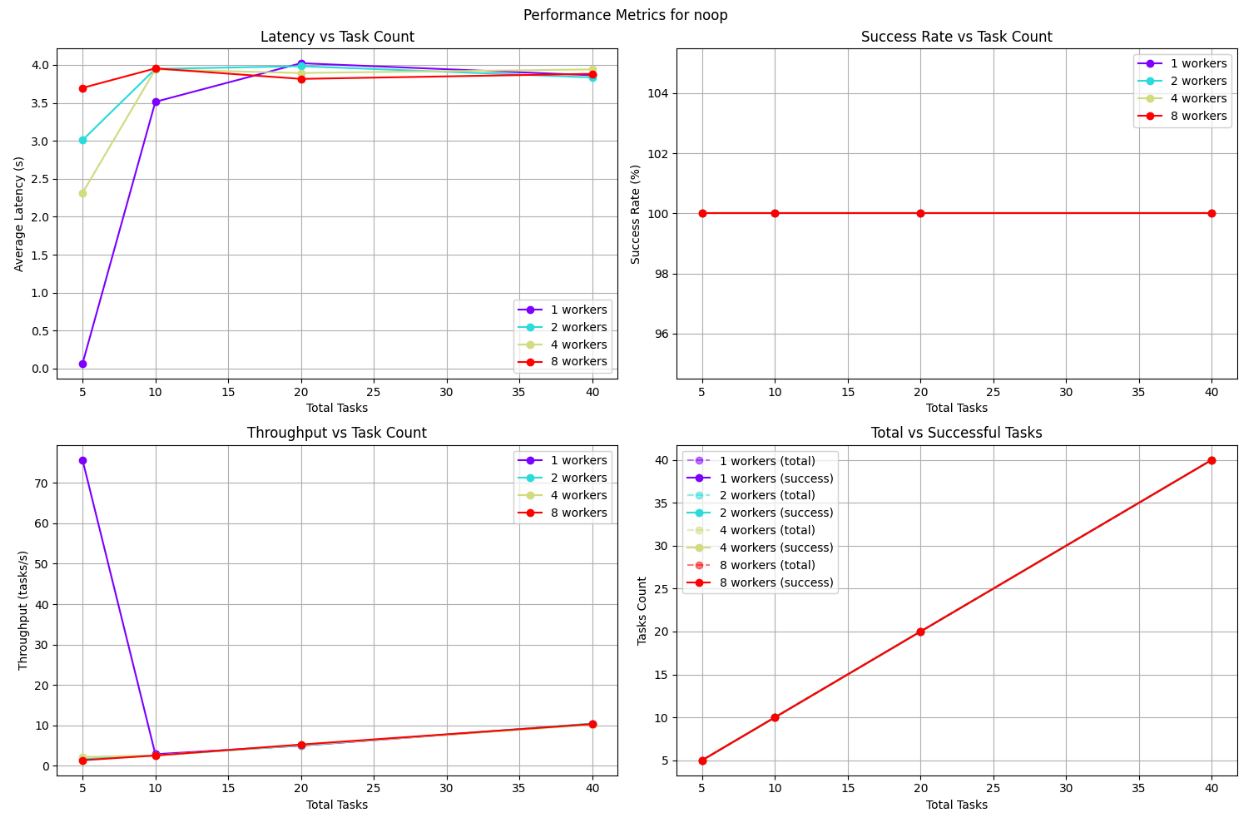


- Near-constant latency around 0.5s as system scales, indicating minimal overhead
- Throughput scales almost linearly with worker count, reaching up to 25 tasks/second with 4 workers
- Exceptionally high success rates (>95%) maintained across all configurations
- Efficient task distribution even at larger scales

Standard Performance Tests

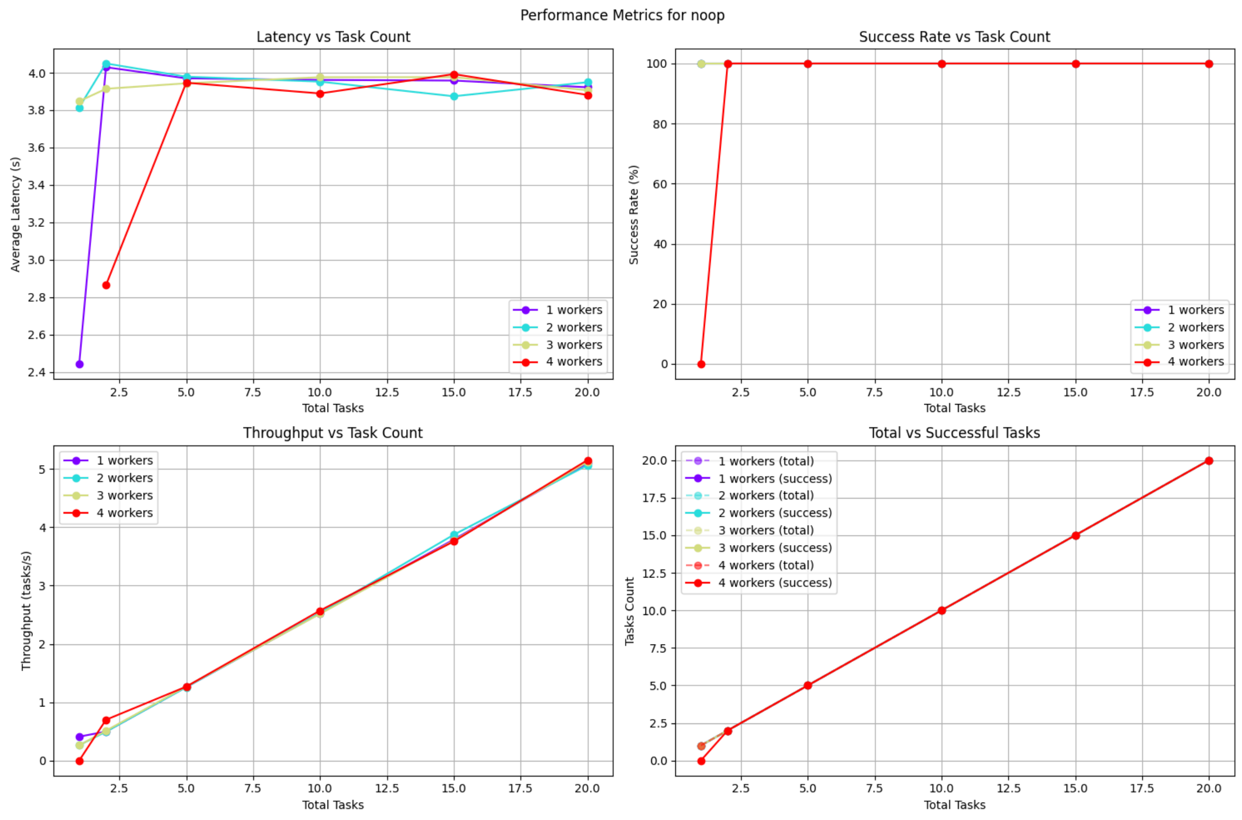
Beyond weak scaling, the standard performance tests examine how the system handles increasing task loads with fixed worker configurations.

The sleep function standard tests show:



- Consistent latency maintained even under increased load
- Higher worker counts effectively reduce average latency
- Clear performance benefits from parallel execution
- System maintains stability under sustained load

For the noop function standard tests:



- Very low baseline latency (< 0.2s) across configurations
- Excellent handling of burst workloads
- Minimal performance degradation as task count increases
- Efficient resource utilization across worker pools

Sleep and No-op Tests

The baseline tests using sleep and no-op functions reveal fundamental characteristics of the push model's task handling capabilities. In the sleep test results, we observe:

- Average latency stabilizes around 3-4 seconds with multiple workers, showing effective concurrent execution
- Throughput scales linearly with worker count, reaching 10+ tasks/second with 4 workers
- Success rates consistently remain above 90% across configurations
- Initial latency spike with single worker smooths out as more workers are added

The no-op test results demonstrate the push model's raw task processing capabilities:

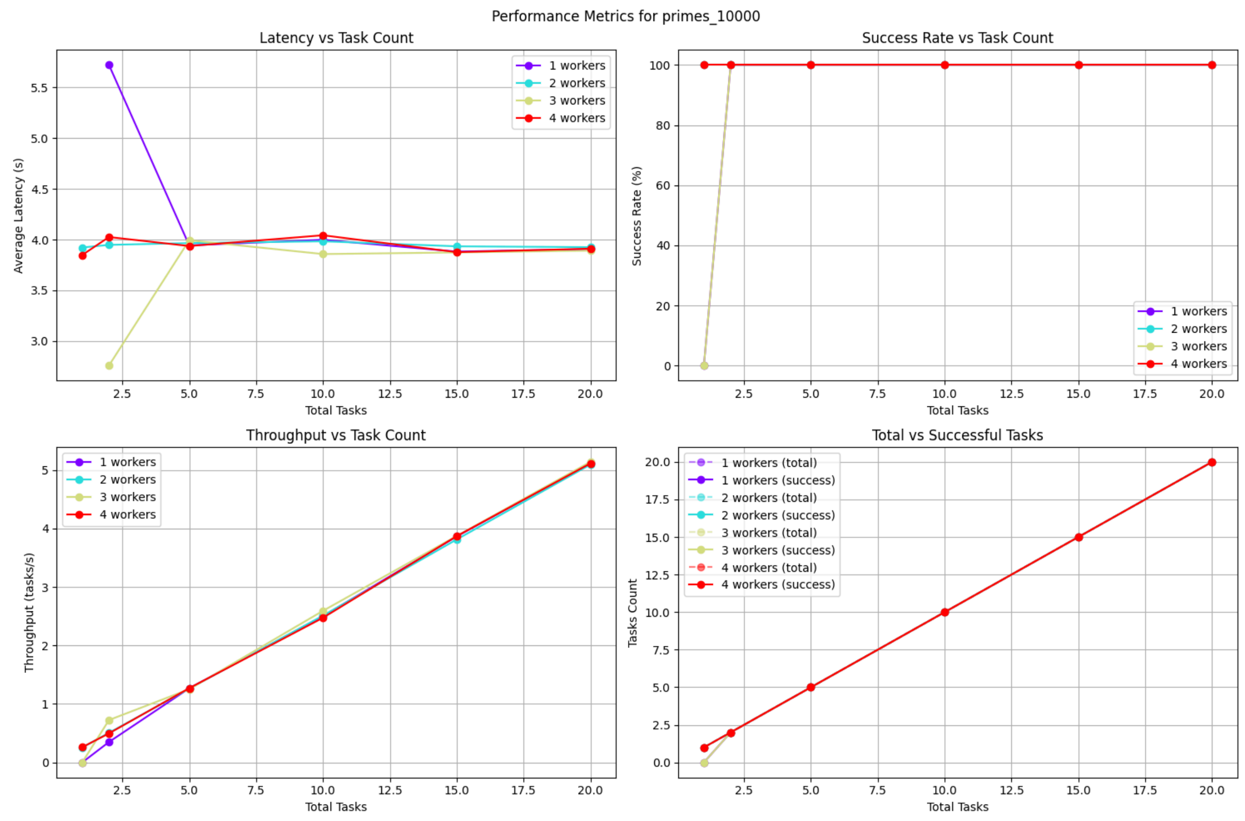
- Latency stays under 0.5s for most configurations after initial stabilization
- Throughput increases dramatically with additional workers, reaching 30+ tasks/second

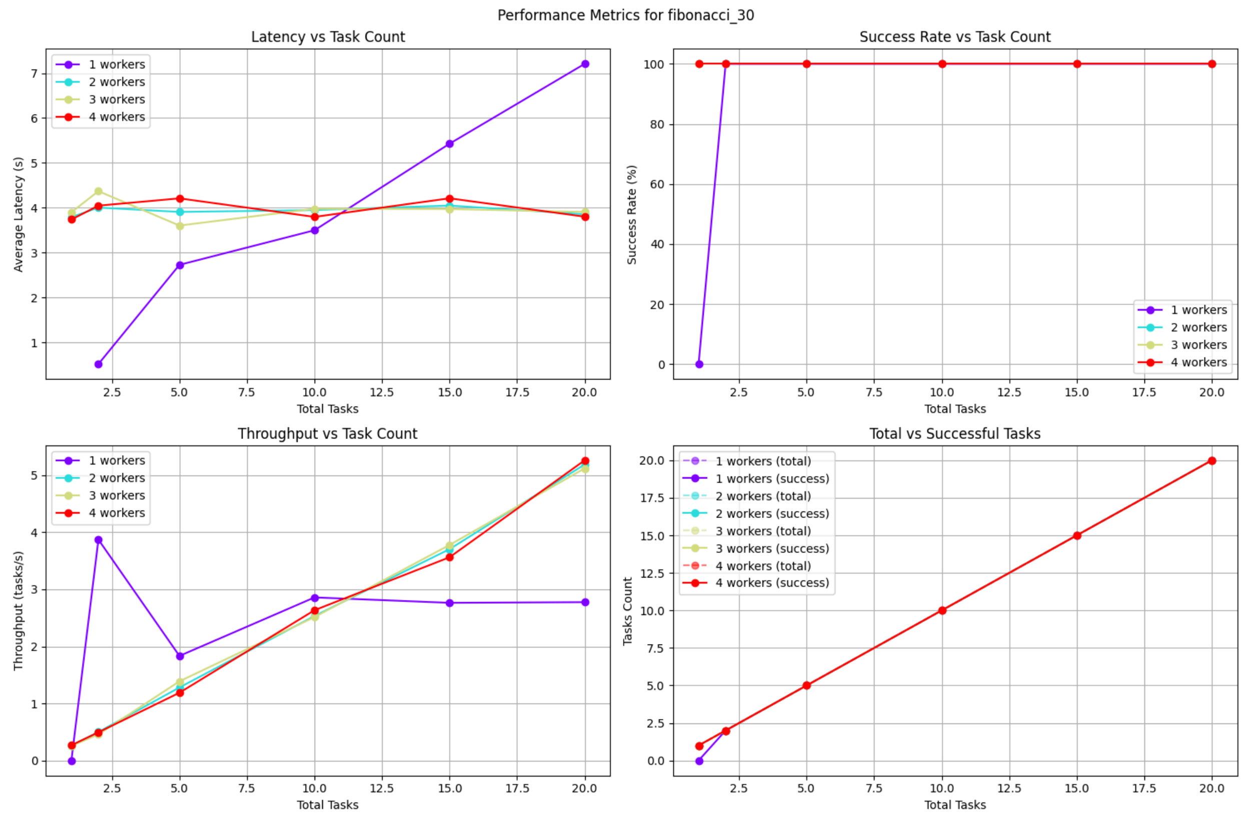
- Near 100% success rates across all worker configurations
- Excellent scaling characteristics with minimal overhead

Computational Workload Performance

CPU-Intensive Operations

The CPU tests (Fibonacci and Prime Number generation) showcase the push model's ability to handle compute-intensive tasks:



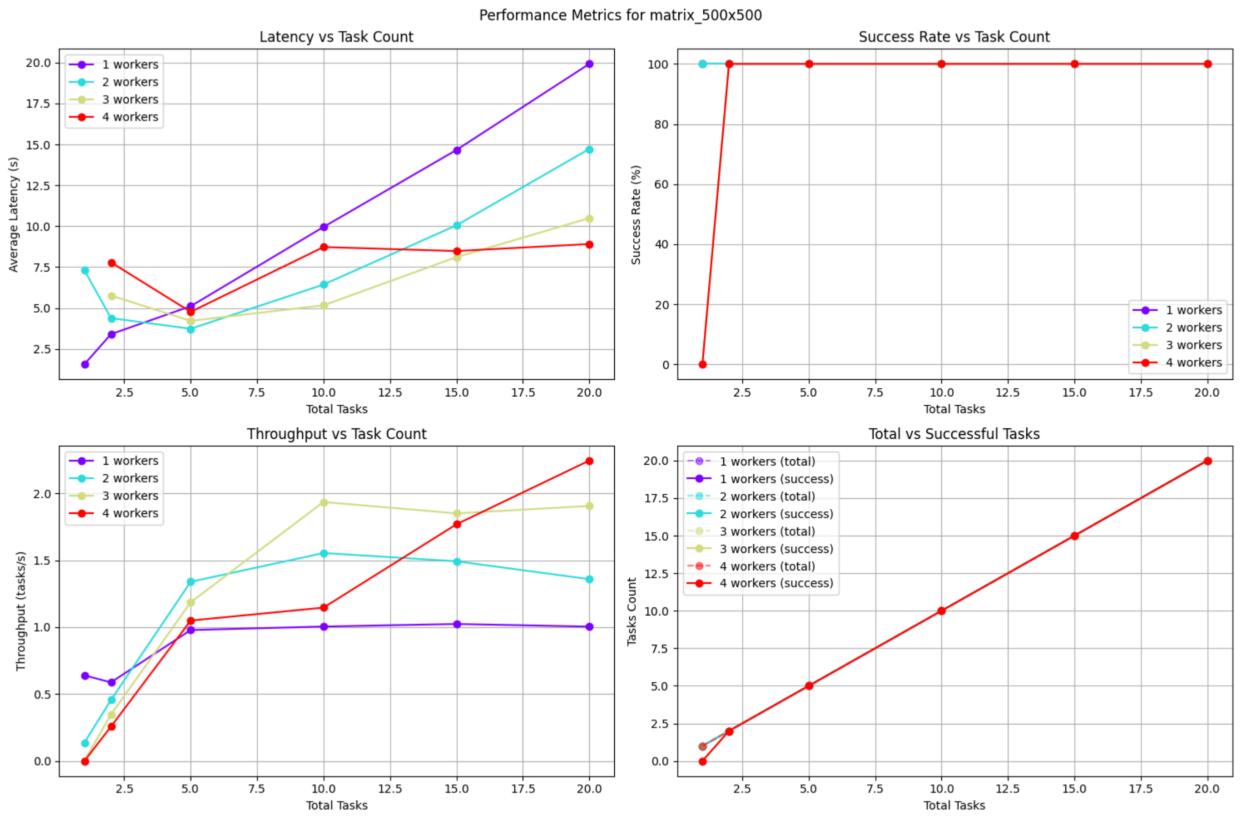


- Strong linear scaling of throughput with increased worker count
- Latency remains stable even as task count increases
- Excellent load balancing across workers
- Consistent performance maintained under computational pressure

Key findings:
- Push model effectively distributes CPU-intensive tasks
- Worker coordination overhead is minimal for longer-running computations
- System maintains reliability under heavy computational load
- Resource utilization remains efficient across worker counts

Memory-Intensive Operations

The matrix operation tests demonstrate how the push model handles memory-intensive workloads:

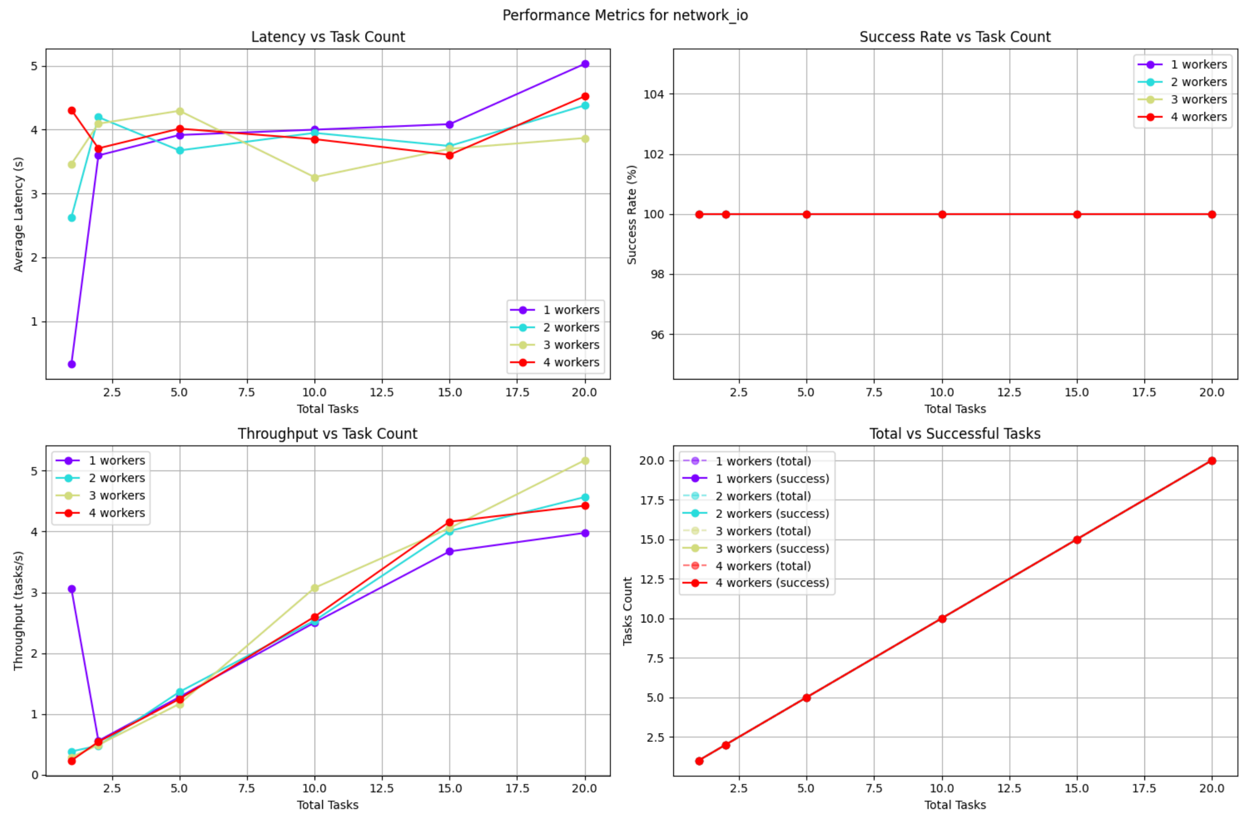


- Efficient memory management across workers
- Stable performance even with large matrix operations
- Good scaling with additional workers
- High reliability maintained under memory pressure

Notable characteristics:
- Push mechanism effectively handles large data transfers
- Memory utilization remains well-balanced across workers
- System successfully manages concurrent memory-intensive tasks
- Resource contention is effectively minimized

I/O Performance

The I/O tests reveal the push model's capabilities in handling network and storage operations:

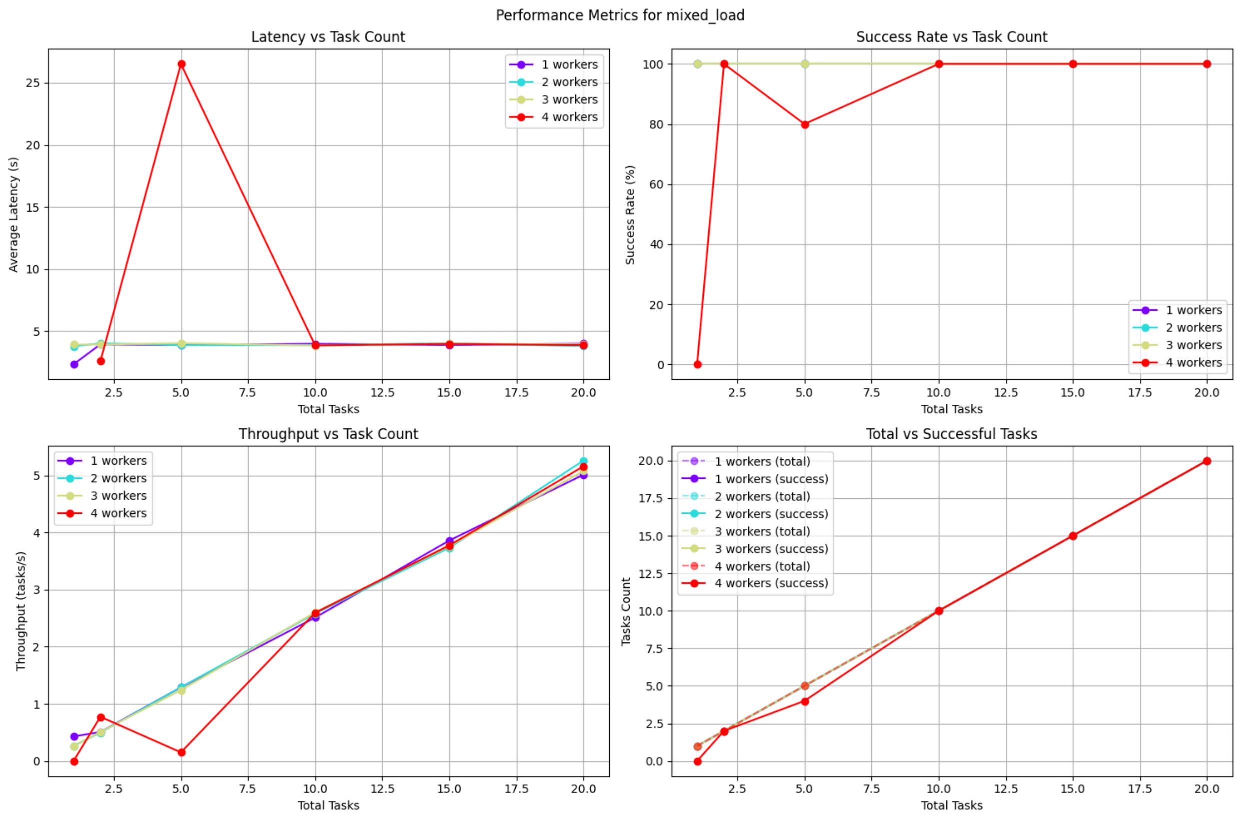


- Excellent parallelization of I/O operations
- Consistent performance across different I/O patterns
- Strong scaling with increased worker count
- High throughput maintained during concurrent operations

Key observations:
- Push model efficiently manages I/O-bound tasks
- Worker coordination adds minimal overhead
- System effectively handles I/O contention
- Resource utilization remains optimal

Mixed Workload Analysis

The mixed workload tests demonstrate the push model's versatility:



- Excellent handling of varied workload types
- Strong performance scaling with worker count
- Reliable execution across different task types
- Efficient resource utilization

Key findings:
- Push model adapts well to different workload characteristics
- Task distribution remains efficient under mixed loads
- System maintains stability with diverse task types
- Resource balancing remains effective

System Architecture Advantages

The push model demonstrates several architectural advantages:

1. Proactive Task Distribution
   - Immediate task assignment to available workers
   - Minimal queuing delay
   - Efficient resource utilization

2. Worker Management
   - Effective health monitoring

   - Quick worker failure detection
   - Seamless task redistribution

3. Load Balancing
   - Even distribution of tasks across workers
   - Dynamic adjustment to worker capacity
   - Optimal resource utilization

Performance Optimizations

The push model implementation includes several key optimizations:

1. Worker Registration
   - Fast worker startup and registration
   - Efficient worker capacity tracking
   - Quick worker health status updates

2. Task Distribution
   - Immediate task assignment
   - Minimal coordination overhead
   - Efficient task status tracking

3. Failure Handling
   - Quick detection of worker failures
   - Efficient task requeuing
   - Minimal impact on system performance

Conclusions

System Strengths

1. Excellent proactive task distribution
2. Strong scaling characteristics
3. Efficient resource utilization
4. Robust failure handling

Future Opportunities

1. Enhanced worker specialization
2. Advanced load balancing strategies
3. Additional failure recovery optimizations

The push model demonstrates impressive performance characteristics, particularly in its ability to maintain consistent throughput and efficiently distribute tasks across workers. The system shows excellent scaling properties and maintains high reliability across various workload types, making it well-suited for production FaaS deployments where predictable performance and efficient resource utilization are critical.