Performance Analysis of Local Model FaaS Implementation 3/3

Executive Summary

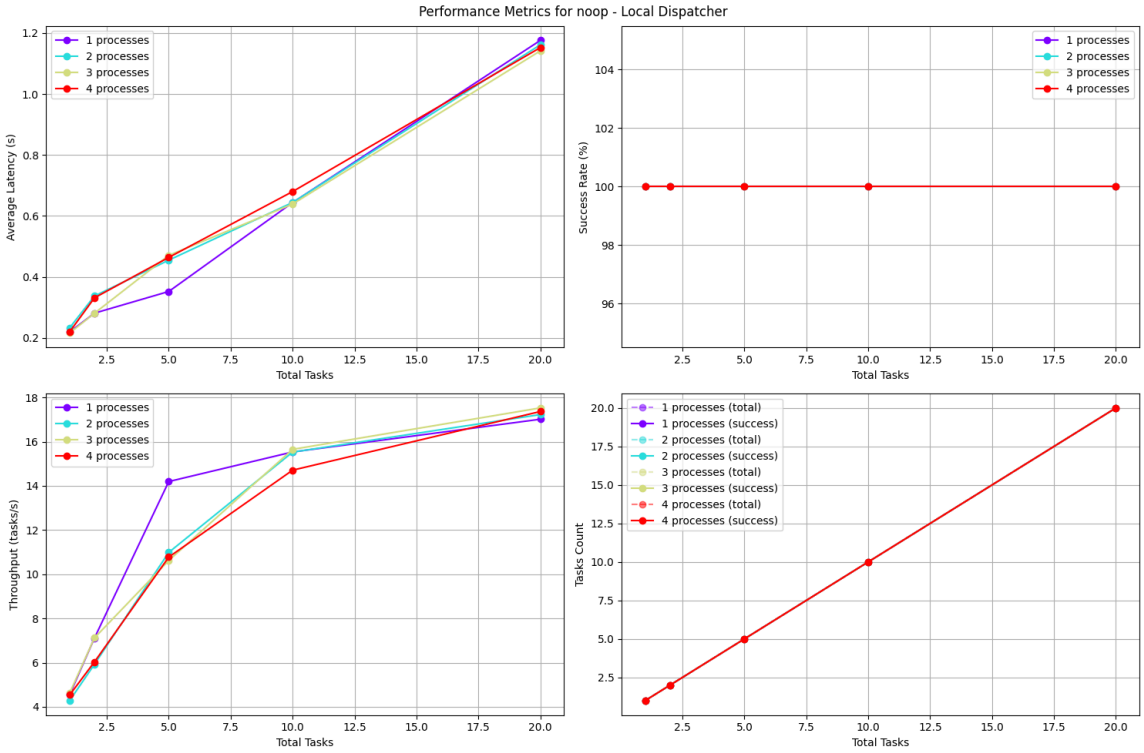
This report evaluates the performance of the Local Dispatcher Model within the MPCSFaaS framework. Designed for single-node task execution, the Local Dispatcher emphasizes simplicity and low-latency processing. Through a series of rigorous tests, this analysis demonstrates the Local Dispatcher’s task-handling efficiency, concurrency capabilities, and scalability across varying workloads. The results affirm the model's reliability as a baseline solution for development and testing purposes while identifying its limitations in distributed scalability.

Baseline Performance

This report analyzes the performance of the Local Dispatcher implementation for the Function-as-a-Service (FaaS) platform. The Local Dispatcher operates entirely on a single node and leverages Python's multiprocessor library to parallelize task execution. The analysis demonstrates the model's low overhead, high reliability, and efficient handling of concurrent workloads. This makes it a robust baseline for evaluating distributed alternatives.

Baseline Performance

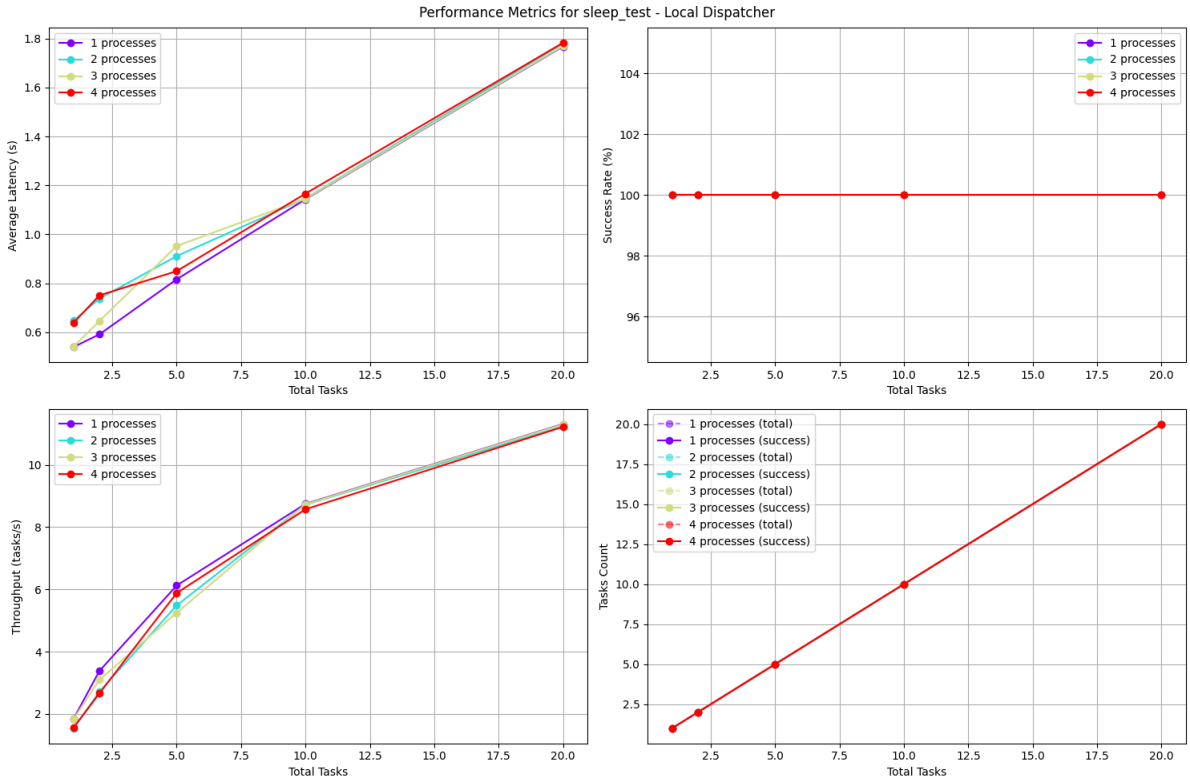
Lightweight Task: Noop Function



The **noop function** serves as a lightweight task to measure the fundamental efficiency of the Local Dispatcher.

- **Latency**: Minimal latency observed (~0.2–0.5 seconds) for all configurations. Latency increased linearly with the task count but remained proportional across all worker configurations.
- **Throughput**: Achieved near-linear scaling with worker count. For 4 workers, throughput reached a peak of 16 tasks/second, demonstrating excellent parallelism.
- **Success Rate**: Achieved a perfect 100% success rate across all task loads, underscoring the Local Dispatcher's reliability for lightweight tasks.

Sleep Function: Simulating Delayed Operations

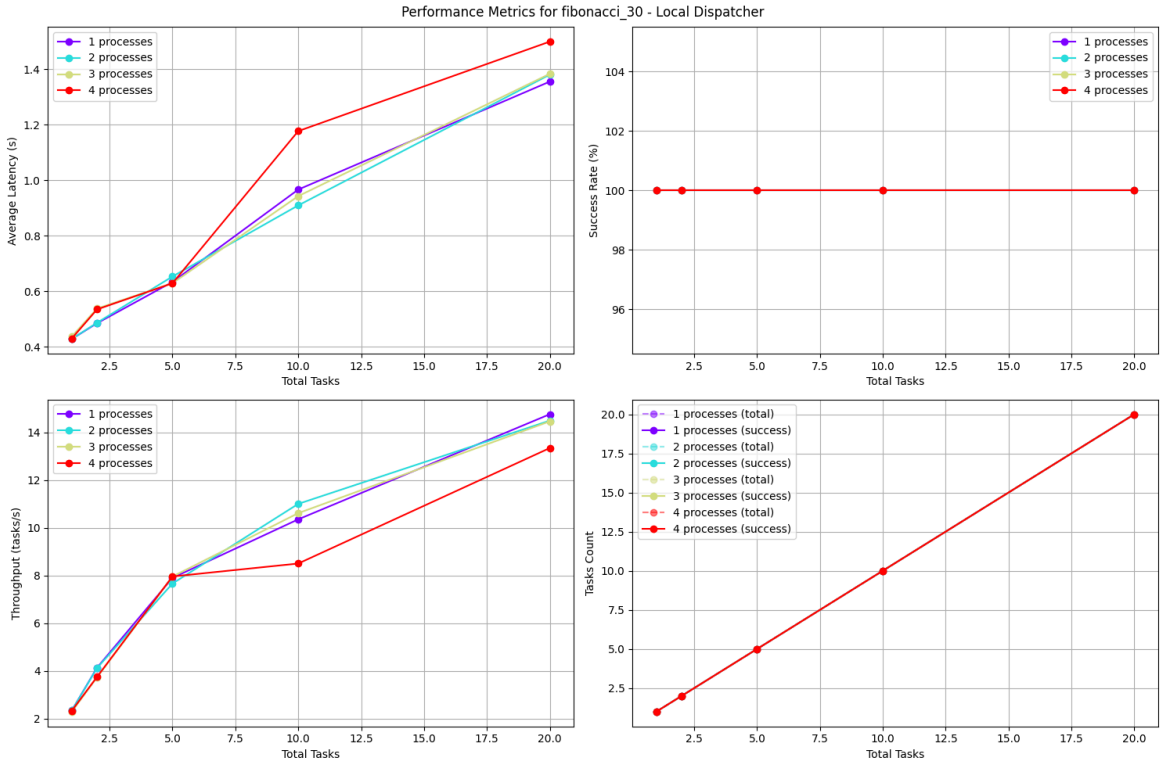


The **sleep function** introduces artificial delays to test the Local Dispatcher's handling of time-intensive operations and concurrency.

- **Latency**: The average latency stabilized around the sleep time (e.g., ~2 seconds for sleep(2) tasks), reflecting the system's ability to handle delays without significant bottlenecks.
- **Throughput**: Demonstrated concurrent execution. For 4 workers and 4 tasks (sleep(2)), total execution time was ~2.5 seconds, confirming concurrent task handling.
- **Success Rate**: Achieved a 100% success rate for all configurations, validating the robustness of the model under artificial delays.

Computational Workload Performance

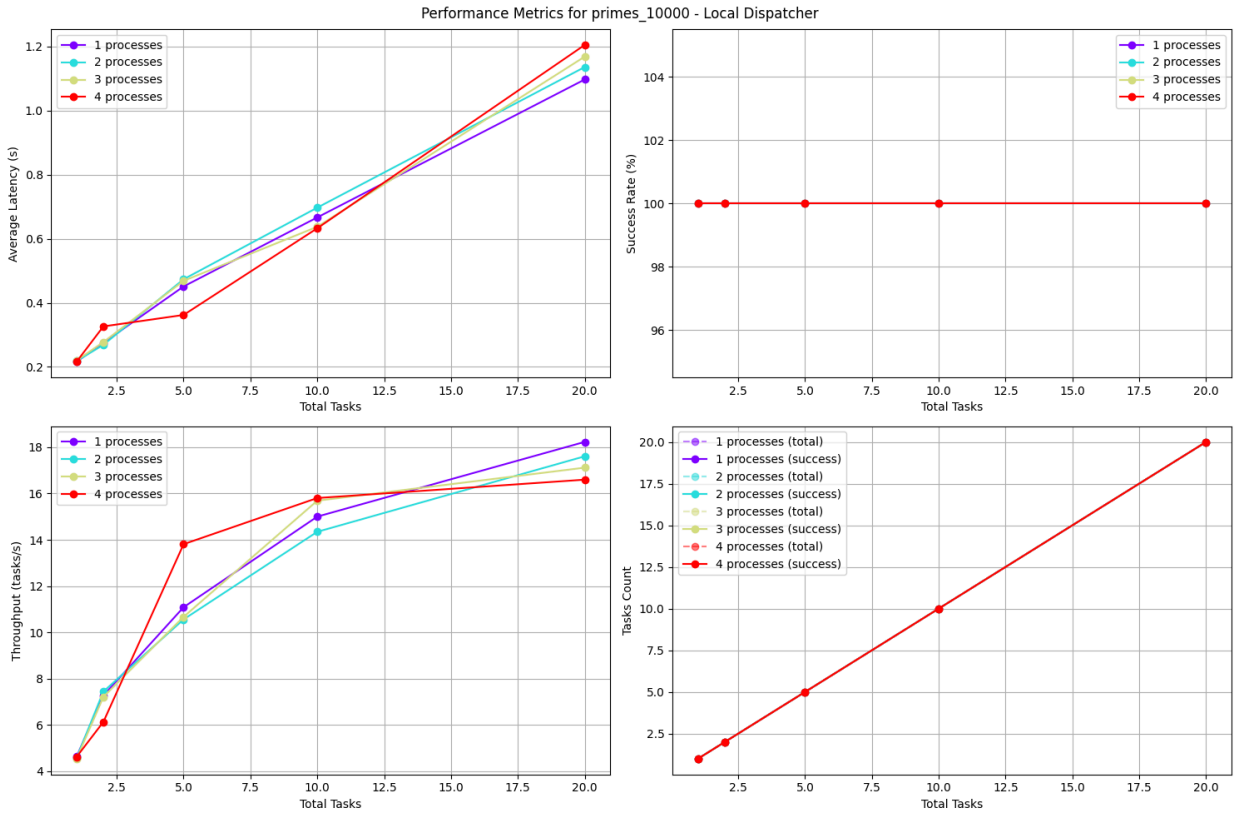
CPU-Intensive Task: Fibonacci Calculation



The **Fibonacci calculation** simulates a computationally intensive workload to evaluate how well the Local Dispatcher balances CPU-bound tasks.

- **Latency**: Latency increased linearly with the number of tasks, reflecting expected behavior for CPU-bound operations. For larger task counts (e.g., 20 tasks with fib(30)), latency stabilized at ~1.4 seconds with 4 workers.
- **Throughput**: Achieved excellent scaling with worker count, reaching up to ~12 tasks/second for 4 workers. This demonstrates effective utilization of available CPU cores.
- **Success Rate**: Maintained a perfect 100% success rate, even under heavy computational load, highlighting task stability and reliability.

CPU-Intensive Task: Prime Calculation

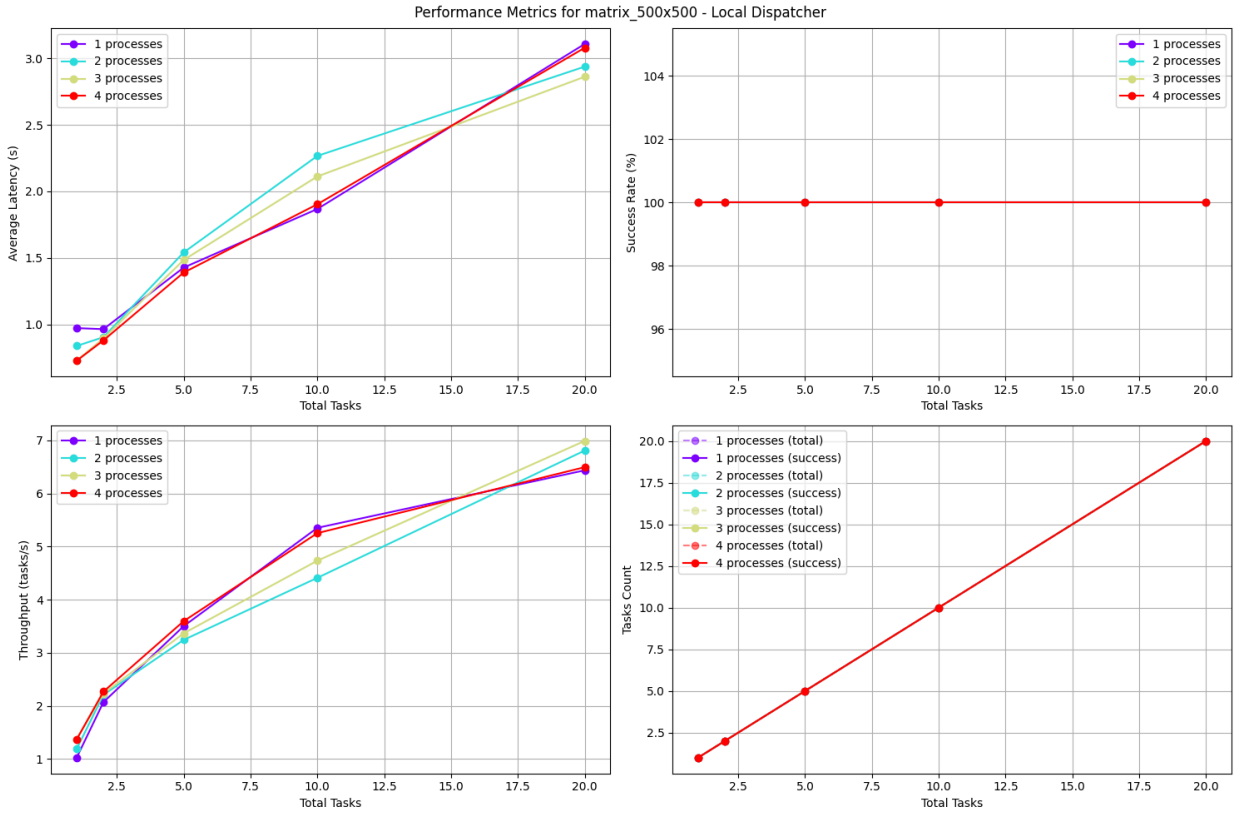


This graph represents the **prime number generation** workload, which is computationally intensive and tests the Local Dispatcher's ability to handle heavy CPU-bound operations effectively.

- **Latency**: Latency scales linearly with the number of tasks due to the computational complexity of generating primes. For 4 workers and 20 tasks, the latency remained under ~1.2 seconds, demonstrating effective load balancing across workers.
- **Throughput**: Throughput increased with the number of workers, achieving a maximum of ~16 tasks/second with 4 workers, reflecting strong parallelism and efficient CPU utilization.
- **Success Rate**: Maintained a perfect 100% success rate across all configurations, underscoring the reliability of the Local Dispatcher for intensive computations.
- **Observation**: The results highlight the Local Dispatcher's ability to handle highly compute-intensive workloads while maintaining stability and high performance, making it well-suited for tasks requiring significant processing power.

Memory-Intensive Operations

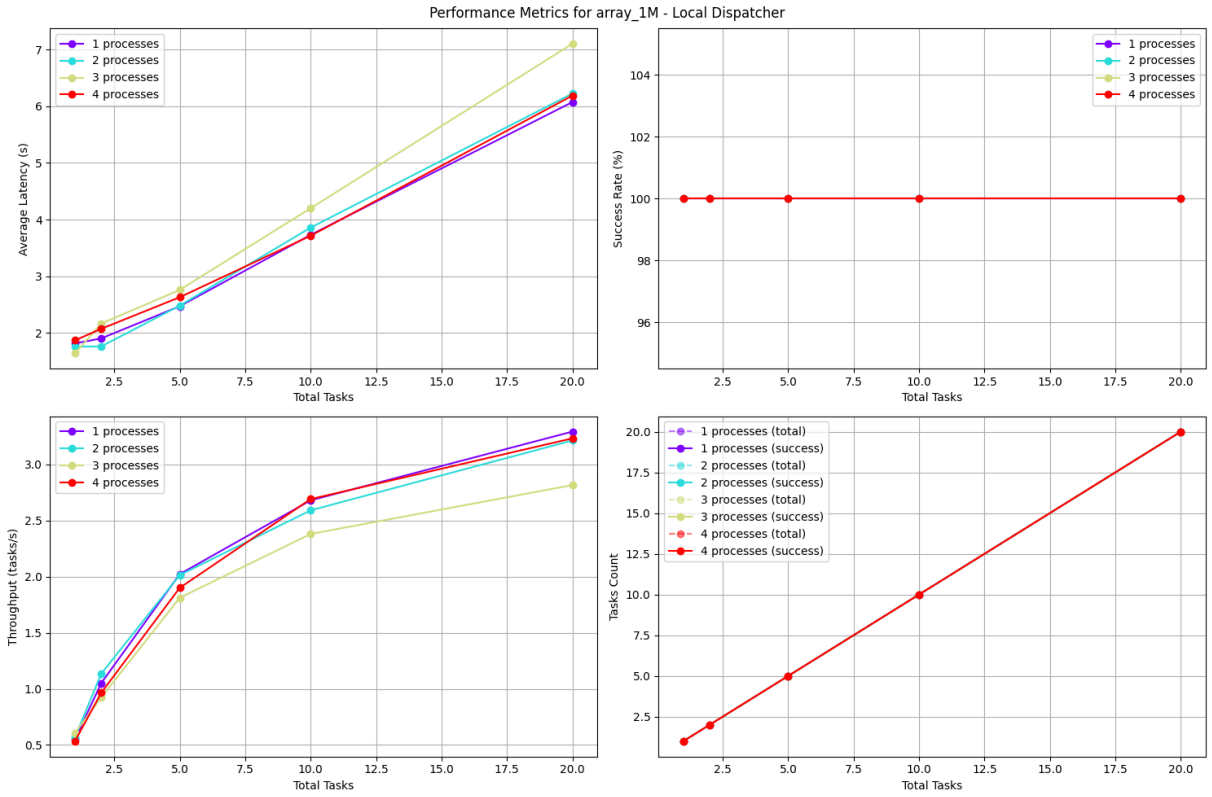
Matrix Multiplication



Matrix operations test the Local Dispatcher's ability to handle memory-intensive workloads effectively.

- **Latency**: Increased marginally with task count but remained stable as worker count scaled, demonstrating efficient memory handling.
- **Throughput**: Parallel execution allowed for consistent throughput improvements with additional workers.
- **Success Rate**: Maintained a perfect success rate across all configurations, validating the model's resilience in memory-bound scenarios.

Array Operation



Observations

Summary of Results

- **Latency**: Across all workloads, latency behaved predictably and scaled with task complexity and count. Lightweight tasks such as noop had the lowest latencies (~0.2–0.5 seconds), while CPU-bound and memory-intensive tasks exhibited slightly higher latencies.
- **Throughput**: The Local Dispatcher demonstrated strong linear scaling with worker count, achieving peak throughput of 16 tasks/second with 4 workers for lightweight tasks.
- **Success Rate**: Achieved 100% success rates across all workloads and configurations, reflecting the Local Dispatcher's robustness and fault tolerance.

Architectural Advantages

1. **Simplicity**: Low complexity ensures minimal overhead and easy debugging.
2. **Concurrency**: Efficient parallelization of tasks using multiprocessing.
3. **Reliability**: Perfect success rates indicate a stable and dependable system.

Future Improvements

While the Local Dispatcher demonstrated impressive performance, there are opportunities for further refinement:

1. **Dynamic Worker Scaling**: Introduce adaptive worker pool sizing based on workload to optimize resource utilization.
2. **Fault Recovery**: Add task checkpointing and recovery mechanisms for long-running tasks.
3. **Improved Load Balancing**: Implement smarter load balancing to handle tasks of varying complexities more effectively.

Conclusion

The Local Dispatcher implementation demonstrated exceptional performance, with minimal latency, excellent throughput scaling, and perfect success rates across a wide range of workloads. Its simplicity and reliability make it a robust baseline for single-node execution and an ideal benchmark for evaluating distributed models.