ecs251

February 2026

1 Evaluation

Our first step is to implement two thread-pool methods: a static and a dynamic thread pool with C++. To compare them fairly, we will benchmark under three representative workload patterns, each defined by a task arrival process that reflects common real-world traffic conditions:

1 Steady load: a constant arrival rate of 1,000 tasks/second for the full duration of the run.

2 Flash crowd: a burst of 5,000 tasks/second followed by a 10-second idle period, repeated in cycles.

3 Daily ramp: alternating phases on 100 tasks/second and 2,000 tasks/second in repeated cycles.

Across these workloads, we will evaluate two resource dimensions that influence the trade-off between static and dynamic thread pools: latency (time cost) and memory (space cost). To isolate how task characteristics interact with thread-pool policy, we will apply two controlled micro-benchmarks:

- Memory-based tasks: $N \times N$ matrix multiplications, where N is tunable to scale memory pressure.
- Latency-based tasks: a synthetic task that blocks for a configurable sleep duration T to emulate downstream I/O or service latency.

We evaluate four task categories for both benchmarks. For each benchmark, we run (1) Light, (2) Medium, and (3) Heavy tasks by increasing the corresponding tuning parameter values (T for latency-based tasks and N for memory-based tasks). Specifically, each request for the latency-based tasks performs a blocking sleep with $T \in \{100\,\mu s, 1\,ms, 10\,ms\}$, to represent the Light, Medium, and Heavy service time. For memory-based tasks, we use $N \times N$ matrix multiplication with $N \in \{128, 256, 512\}$ to represent Light, Medium, and Heavy working-set sizes, respectively. Finally, we include (4) a Mixed workload for each constraint to capture heterogeneity: 60% Light / 30% Medium / 10% Heavy, where each request samples its T or N from the setting according to this

Metric	**Definition**
Throughput	Tasks completed per second
Latency (p50, p95, p99)	Time from task submission to completion
PSS Memory	Proportional Set Size (attributable RAM used by the process)
Thread Count	Active threads over time

Table 1: Metrics collected in each benchmark run.

distribution.

For each benchmark run, we will record four metrics to evaluate the time and memory costs accodingly (Table 1).

We will run 12 experiment configurations in total: three workload patterns $\times$ 2 pool types (static and dynamic) $\times$ 2 core counts (4 cores and 16 cores). We will fix the CPU resources available to the benchmark by restricting it to exactly 4 or 16 cores using Linux CPU affinity (e.g., tasket). For each configuration, we run the benchmark for 120 seconds and repeat the process three times; we then aggregate the metrics across the three trials and report the mean and variance.