Ecs251 project proposal

Harshil Patel[1]
[1]Department of Computer Science, University of California, Davis

1 Introduction & Motivation

Modern concurrent systems have to choose between static thread pools (pre-created threads) and dynamic thread pools (threads created on demand), but they lack the empirical data to make informed decisions. Static thread pools waste memory during low load and idle periods which results in a constant overhead regardless of the workload. Dynamic thread pools avoid this waste by creating threads only when needed, but they incur overhead for thread creation and destruction, which can lead to performance degradation under high load. In this project, we want to evaluate the performance trade-offs between static and dynamic thread pools under varying workloads and system configurations and answer the question: At what point does the overhead of dynamic thread creation outweigh its memory savings compared to static thread pools?

This tradeoff can directly impact the performance and costs of cloud-based systems (Freire et al., 2021). A web service handling variable traffic has to decide between a static pool with constant resource usage or a dynamic pool that adapts to demand but may suffer from latency spikes during high traffic. As cloud providers charge based on resource usage, understanding this tradeoff can lead to significant cost savings while maintaining performance. Apart from costs, this question also affects other aspects of system design such as database connection pools, task scheduling in distributed systems, etc which all use thread pools to manage concurrency. Yet, there is a lack of comprehensive studies that quantify the performance implications of static vs dynamic thread pools across different workloads and system configurations. By systematically evaluating these trade-offs, we can provide practical guidelines for system architects to choose the right thread pool strategy based on their specific workload and performance requirements.

2 Background & Related Work

Existing research mainly focuses on optimizing thread pool sizes rather than comparing different thread pool archtectures. For example, Lee et al. (Lee et al., 2011) propose Trendy Exponential Moving Average (TEMA), a predictive model that adjusts the thread pool size based on workload patterns and system metrics to optimize performance. Costa et al. (Costa et al., 2019) introduce ADAPT-T, an adaptive thread pool that dynamically tunes its size based on real-time performance metrics. Other works examine thread pool configurations. For example, Freire et al. (Freire et al., 2021) compare global thread pools (single pool and queue for all tasks) against local thread pools (each task has its own dedicated pool and queue). Their study simulates these configurations under high workloads to demonstrate that optimzed local pools result in a lower average makespan when compared to global pools. While Freire et al. (Freire et al., 2021) provide insights into thread pool configurations, they focus on optimal distribution of a fixed number of threads, while our work aims to compare static vs dynamic thread pool architectures across varying workloads and system settings.

References

Nilushan Costa, Malith Jayasinghe, Ajantha Atukorale, Supun Abeysinghe, Srinath Perera, and Isuru Perera. 2019. Adapt-t: An adaptive algorithm for auto-tuning worker thread pool size in application servers. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6.

Daniela L Freire, Angela Mazzonetto, Rafael Z Frantz, Fabricia Roos-Frantz, Sandro Sawicki, and Vitor Basto-Fernandes. 2021. Performance evaluation of thread pool configurations in the run-time systems of integration platforms. *International Journal of Business Process Integration and Management*, 10(3-4):318–329.

Kang-Lyul Lee, Hong Nhat Pham, Hee-seong Kim, Hee Yong Youn, and Ohyoung Song. 2011. A novel predictive and self – adaptive dynamic thread pool management. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*, pages 93–98.