Ecs251 project proposal

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

1 Introduction & Motivation

The advancements in multicore architecture and multiprocessor computers are increasing and ongoing (Adam, 2022). To keep up with this trend, we need techniques like multithreading to maximize the performance of ever-advancing cpu's (ni). Multithreading is of vital importance for this because it can harness the power of multiprocessor computer, improve system reliability by preventing one operation from negatively affecting another in a program(user interface event affecting time-critical operation), and maximize cpu use (in programs that read/write from a file, perform I/O, or poll the user interface) (ni).

While a very efficient tool for harnessing the power of multiprocessor computer, multithreading does come with some challenges. The top most important challenge in multithreading is the management of thread pools (Lee et al., 2011), namely the size of the thread pool. The number of threads in a thread pool determine the changes in response times and resource utilization (Lee et al., 2011). In our research, we aim to collect empirical data that informs decisions on thread pool size. In doing so, we also aim to propose a “dynamic thread pool” with varying number of threads such that we can optimize minimum response time for maximum resource utilization. We will evaluate 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?

2 Background & Related Work

Current heuristics that are used to determine the number of threads are flawed (Ling et al., 2000). In our preliminary research, we find that there have been few researches on dynamic model for thread size. One of the first such model was the Watermark model which adjusts the number of threads according to the current number of incoming requests (Kim et al., 2007). The issue with this model is that while it solves the problem of “efficient usage of resources”, it doesn't efficiently obtain the required number of threads in advance (Kang et al., 2008). Furthermore, there are also prediction based models to determine thread pool size, but in such models, only factors such as the request rate or number of worker threads is considered in the management (Lee et al., 2011). One of the research that has proposed dynamic thread pool model(prediction based), themselves point out that the model lacks testing for a longer period of time and on general server environment which provides various services simultaneously (Ling et al., 2000). Another research adds to the prediction model by proposing what they call the "Trendy exponential moving average model" and again the issue with this model seems to be that it doesn't have the desired accuracy in its prediction (Lee et al., 2011).

Of the many researches into dynamic sizes of pool, none look at the memory usage/CPU overheads and whether those are significantly lower than static pool to signify the performance cost. The trade off for replacing static threads affect the performance and costs of cloud-based systems (Freire et al., 2021), use of dynamic pools in web services handling could have latency spikes during high traffic. Not just costs and such latency issues, there is also the question of system design such as database connection pools, task scheduling in distributed systems, etc which all use thread pools to manage concurrency. 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.

example citation (Hu et al., 2024).

References

[link].

George K Adam. 2022. Co-design of multicore hardware and multithreaded software for thread performance assessment on an fpga. *Computers*, 11(5):76.

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.

Hanxu Hu, Simon Yu, Pinzhen Chen, and Edoardo M Ponti. 2024. Fine-tuning large language models with sequential instructions. *arXiv preprint arXiv:2403.07794*.

DongHyun Kang, Saeyoung Han, SeoHee Yoo, and Sungyong Park. 2008. Prediction-based dynamic thread pool scheme for efficient resource usage. In *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, pages 159–164. IEEE.

Ji Hoon Kim, Seungwok Han, Hyun Ko, and Hee Yong Youn. 2007. Prediction-based dynamic thread pool management of agent platform for ubiquitous computing. In *International Conference on Ubiquitous Intelligence and Computing*, pages 1098–1107. Springer.

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. IEEE.

Yibei Ling, Tracy Mullen, and Xiaola Lin. 2000. Analysis of optimal thread pool size. *ACM SIGOPS Operating Systems Review*, 34(2):42–55.