A Brief Review of “Capriccio: Scalable Threads for Internet Services”

Yi Luo

1 Summary

This paper introduces a scalable thread-based package named Capriccio for high-concurrency Internet services. It claims that the current popular event-based system has a bunch of drawbacks. For example, event-based systems hide the control flow through applications so that they are difficult to understand especially when examining and debugging the code. On the contrary, thread-based systems provide natural and intuitive abstraction for the programming. Instead of improving event-based systems, this paper decides to ”fix” the thread-based model.
The Capriccio system adopts user-level threads which provides with much more benefits than weakness. By decoupling the thread package implementation from the underlying operating system, the Capriccio system is able to exploits new I/O mechanisms and compiler supports to realize techniques such as linked stacks and resource-aware scheduling so as to achieve dramatical scalability and performance improvements when compared to event-based and existing thread-based systems. [1]

2 Merits

I like the idea of dynamic stack allocation of Capriccio. According to my Java programming experience, the stack in JVM is bounded and is allocated in a consecutive space. In Capriccio’s implementation, the idea of linked state management is employed. Based on the compiler feature, the system estimates the space to allocate by considering the call graph and a number of checkpoints placed on it. For example, in a basic call graph which contains no recursive cycles, the space estimated by the MaxPath can be used as the desired stack space. In addition, because stack chunks are managed in the linked list, the space is typically non-contiguous and can grow or shrink at run time. Experiments given in this paper also verify the good performance and efficiency of such stack implementation.
Then, in the discussion of the related work, the paper mentions another idea that allocates activation records on the heap so that it does not use the stack at all. The paper gives rational explanation why this idea is not considered since it relies on garbage collectors for the heap manipulation, which is not supported by C programming. Also, general-use garbage collectors are usually not very suitable for high-performance systems.
Besides, it is also cool to have a practical case study based on the Apache 2.0.44 server to evaluate the performance of the linked stack management.

3 Drawbacks

However, as mentioned in the future work. The proposed Capriccio in this paper can just deal with a single CPU because it depends on the cooperative threading model to provide atomicity. On multi-CPU machines, the system has to makes use of the information provided by the compiler to guarantee atomicity to a certain degree.

4 Personal Thoughts

Also in the future work, the paper states a potential work to produce profiling tools to assist automate parameter tuning for the stack management. I think this idea is cool. By recording information about wasted space and gathering statistics about function calls, we are able to make use of some machine learning modules to generate mathematical models for the future prediction, which automates the tuning process of parameters.

References

[1] Rob von Behren, Jeremy Condit, Feng Zhou, George C. Necula, Eric Brewer. Capriccio: Scalable Threads for Internet Services. *SOSP*, 2003.