Critique for Capriccio: Scalable Threads for Internet Services

Xin Wang

1 Summary

The paper "Capriccio: Scalable Threads for Internet Services" [2] presents a scalable thread package on high-concurrency servers in order to satisfy the increasing scalability demands. Current comodity hardware has enabled servers to handle tons of simultaneous connections while current software can hardly achieve the same goal without significant performance degradation. Though event-based systems can handle requests using a pipeline of stages, such systems have drawbacks such as hiding control flow which mades examining source code and debugging difficult compared to thread-based systems. The contribution of this paper is to fix the thread-based model to achieve high concurrency performance instead of using event-based one. The goals include supporting existing thread APIs, enable large scalability of threads and flexibility to address application-specific needs. The new thread package Capriccio proposed in this paper helps to improve the scalability of basic thread operations, to solve stack allocation problem for large number of threads by introducing linked stacks and to extract control flow information to make scheduling decisions by a resource-aware scheduler.

Capriccio is a user-level thread package which supports POSIX API. The reason why it chooses user-level threads is that user-level threads can provide flexibility and good performance when such threads greatly reduce the overhead of thread synchronization, avoid requiring kernel crossings for mutex acquisition or release and make memory management more efficient. On the other hand, user-level threads are disadavantagable when retaining control of the processor during a blocking I/O call execution, when introducing a wrapper layer that translates blocking I/O mechanisms to non-blocking I/O ones, and when taking advantage of multiple processors.

Capriccio is implemented on top of Edgar Toernig's coroutine library that provides fast context switches. Capriccio also intercepts blocking I/O calls at the library level, which works flawlessly for both dynamically and statically linked applications. Capriccio provides a modular schedularing mechanism that hides the event-driven behavior from the programmer and allows users to easily select between different schedulers. Capriccio takes advantage of cooperative scheduling in order to improve synchronization.

As to linked stack management, compiler analysis is used to limit preallocated stack space amount. The analysis is based on a weighted call graph, with which a reasonable bound on the amount of stack space consumed by the threads. To implement dynamically-sized stacks, the call graph analysis identifies call sites where checkpoints are inserted. When placing checkpoints, we need a bound on the stack space that should be consumed before our reaching the next checkpoint. Since wasted stack space may happen when a new stack chunk is linked or exist at the bottom of current chunk, users are allowed to tune the algorithm for fewer wated space and faster execution speed. The linked stack algorithm has made preallocation of large stacks unnecessary therefore reduces vitual memory presure; this algorithm also improves paging behavior and reduces overal memory waste.

For resource-aware scheduling, Capriccio provides application-specific scheduling for thread-based applications so that we can use the automated scheduling to control admission and to improve response time. The key abstraction for scheduling is blocking graph. Capriccio generates this graph by observing the transitions between blocking points. Caprissio can learn the application dynamically and improve scheduling and admission control accordingly. Capriccio also introduces resource-aware scheduling to keep track of resource utilization levles and decide if resource reaches the limit, annotate each node with resources to predict the resource's impact, and to prioritize nodes for scheduling.

2 Positive reactions

First, Capriccio has the advantages of user-level threads such as flexibility, being lightweighted, inexpensive synchronization and fast context switches. Second, the most significant advantages of Capriccio are its scalability, efficient stack management and resource-aware scheduling. According to the experimental results, the fixing thread package of Capriccio is a good solution to the problem of establishing scalable and high-concurrency servers. Resource-aware scheduling and linked stacks also successfully make performance improvements and help to achieve scalability compared to event-based and thread-based systems.

3 Negative reactions

First, Capriccio system ensures that threads be made as memory efficient as events-based system. This requires modification to C compiler so that the compiler can perform dynamic memory allocation of stack space. However, this is not quite convenient since dynamic memory allocation may rapidly cause memory fragmentation in memory-constrained devices so that the dynamic allocation of stack memory is not very feasible.

Second, the Capriccio thread package implements resource-aware scheduling by monitoring behavior of the threads. The thread package also measures the resource requirements of the threads. Then scheduling decisions are made accordingly and resource utilization is optimized. Such method is a bit generalized than multithreaded chip multiprocessors as mentioned in [1], since it can optimize for usage of different types of resources, yet requires detailed monitoring of the program state.

4 Extension proposal

Based on the implementation of Capriccio, future programmers can provide more information related to high-level structure of the servers' tasks to compiler. The prorgrams may be written in threaded style. [2]

5 Conclusions

To sum up, Capriccio is a good proposal for high-concurrency servers and has helped to deal with many empirical problems such as scalability and scheduling. It also provides us with good insight about how new techniques can be integrated to compiler.

References

[1] Margo Seltzer, Christopher Small, and Daniel Nussbaum. Performance of multithreaded chip multiprocessors and implications for operating system design, alexandra fedorova. 2005.

[2] Rob Von Behren, Jeremy Condit, Feng Zhou, George C Necula, and Eric Brewer. Capriccio: scalable threads for internet services. *ACM SIGOPS Operating Systems Review*, 37(5):268–281, 2003.