Capriccio: Scalable Threads for Internet Services
Matthew Phillips

Overview
• Motivation
• Background
  • History
  • Threads vs. event based systems
    • User-level threads vs. kernel threads
  • Capriccio
    • Linked stacks
    • Resource-aware scheduler
• Evaluation

Motivation
• High concurrency
  • Internet servers can have requests for hundreds of thousands of connections.
  • Simplify programming model.

Background
• History
  • Need for high concurrency servers
    • Flash
    • SEDA
  • Interest in systems other than thread based systems.
    • Event based systems
  • Problem: Event based systems are sometimes difficult to use.
    • Programmers prefer linear program control
  • Why threads are a bad idea (for most purposes), John Ousterhout. Sun Labs. 1996.
    • Race conditions, deadlocks, not scalable, etc.
  • Popularity of Internet about the same time
    • Heavy demand on web servers

Background
• History
  • Instead of making event based systems more usable, “fix” thread based systems.
  • Investigate ways to make thread based systems more scalable.
  • Authors presented Why events are a bad idea (for high-concurrency servers), 2003
    • On the duality of operating system structures, Lauer, Needham. 1978.
Background

• Why threads?
  • Can have equal or better performance
  • More desirable programming model
    • Programmers prefer linear program control
    • Existing systems use threads

Background

• Kernel threads
  • Systems calls
    • open, read, send, receive
  • User-level threads
    • User-level libraries

Background

• User-level threads
  • Advantages:
    • Flexibility: A level away (abstraction) from kernel. Decouples applications and kernel.
    • Performance: User-level threads are lightweight: Inexpensive synchronization, fast context switches
  • Thus, user-level threading is used.

Background

• Created user-level threading library for Linux
  • All thread operations are O(1)
    • Now scalable

Capriccio

• Linked stacks
  • Traditional call stacks
  • LinuxThreads allocates 2 megabytes per thread
  • Most threads only consume a few kilobytes
  • Lots of wasted virtual memory
    • 32 bit system has 4 gigabyte limit

Capriccio

• Uses weighted call graphs
  • Perform a whole-program analysis
    • GIL used to read source
  • Each function is a node
  • Each edge is a function call
  • Node weights are calculated from stack frames
Capriccio

- Node weights
  ```c
  int sample() {
    int a, b;
    int c[10];
    char e[8036];
    return 0;
  }
  ```

Capriccio

- Example
  ```c
  main(){char buf[512]; A(); C();}
  void A(){char buf[820]; B(); D();}
  void B(){char buf[1024];}
  void C(){char buf[205]; D(); E();}
  void D(){char buf[205];}
  void E(){char buf[205]; C();}
  ```

Capriccio

- Example
  Path from Main->A->B is 2.3k

Capriccio

- Linked stacks
  - Recursion complicates things
  - Use checkpoints to deal with recursion

Capriccio

- Placement of checkpoints
  - Break each cycle with checkpoint
  - Additional checkpoints
    - Longest path from node to checkpoint, if predefined limit is exceeded, add checkpoint

Capriccio

- Example
  ```c
  // Diagram of linked stacks and checkpoints
  ```
Capriccio

• Resource aware scheduling
• Uses blocking graph
  • Node is location in program that is blocked
  • Node is composed of call chain used to reach blocking point

• Evaluation
  • Apache
  • Compared with Knot
  • Compared with Haboob

15% speedup with Capriccio