Why Events Are A Bad Idea

Rob von Behren, Jeremy Condit, and Eric Brewer
University of California at Berkeley

Presented by: Mikael Fernandus Simalango

Summary

- Authors of this paper want to convey their statement that thread-based system is comparable with event-based system in term of achieving highly concurrent applications

Highly Concurrent Apps

- It's hard to build because:
  - Handling large numbers of concurrent task requires the use of scalable data structures
  - These systems typically operate near maximum capacity -> creating resource contention and high sensitivity to scheduling decisions
  - Race conditions and subtle corner cases (problem when parameters are extreme) are common -> debugging and code maintenance becomes difficult

Why Events Are Considered Better Than Threads

- Primary reasons:
  - Inexpensive synchronization due to cooperative multitasking
  - Lower overhead for managing state (no stacks)
  - Better scheduling and locality, based on application-level information
  - More flexible control flow (not just call/return)

Threads vs Events

- Duality by Lauer and Needham

Events	Threads
Event handlers	Monitors
Events accepted by a handler	Functions exported by a module
SendMessage/ Await Reply	Procedure call, or fork/join
SendReply	Return from procedure
Waiting for messages	Waiting on condition variables

Disproval to Thread Criticisms

- Topic: **Performance**
- Criticism: Many attempts to use threads for high concurrency have not performed well
- Counter argument:
  - Existing thread systems are developed in operation with order O(n) ->design flaw
  - Optimized version of Pthreads scales quite well up to 100,000 threads

Disproval to Thread Criticisms (cont’d)

- Topic: **Control Flow**
- Criticism: threads have restrictive control flow
- Counter argument:
  - Control flow for event system, except dynamic fan-in and fan-out, falls into 3 categories: call/return, parallel calls, pipelines which can be expressed more naturally with threads
  - Existing event system also doesn’t use complex pattern for control flow

Disproval to Thread Criticisms (cont’d)

- Topic: **Synchronization**
- Criticism: Thread synchronization mechanisms are too heavyweight
- Counter argument:
  - Adya et al show that ease in event synchronization us is really due to cooperative multitasking, not events themselves -> cooperative thread system can also reap the same benefits

Disproval to Thread Criticisms (cont'd)

- Topic: **State Management**
- Criticism: thread stacks are an ineffective way to manage live state -> tradeoff between risking stack overflow and wasting virtual address space on large stacks
- Counter argument:
  - A proposal for a mechanism that will enable dynamic stack growth

Disproval to Thread Criticisms (cont'd)

- Topic: **Scheduling**
- Criticism: The virtual processor model provided by threads forces the runtime system to be too generic and prevents it from making optimal scheduling decisions
- Counter argument:
  - Lauer-Needham duality indicates that scheduling tricks to cooperatively schedule threads can also be applied at application level

Why Threads Fit Better For High Concurrency

- Authors’ claims:
  - Topic: control flow
    - Event-based programming tends to obfuscate control flow of the application
    - Thread systems allow programmers to express control flow and encapsulate state in a more natural manner
  - Topic: exception handling and state lifetime
    - Cleaning up task state after exceptions and after normal termination is simpler in threaded system because the thread stack naturally tracks the live state for that task
    - In event systems, task state is typically heap allocated -> rely on garbage collection

Why Threads Fit Better For High Concurrency (cont’d)

- Authors’ claims (cont’d):
  - Topic: existing systems
    - Even event-driven systems subtly prefer threads
    - Thread systems are simpler to build, especially for non highly concurrent system -> scale to high concurrency
  - Topic: just fix events
    - Fixing the problem with events requires more effort than switching to threads

Compiler Support for Threads

- Modification to compiler to support highly-concurrent thread systems:
  - Dynamic stack growth
    - Allowing the size of the stack to be adjusted at run time through compiler analysis
  - Live state management
    - Reordering variables with overlapping lifetimes in order to prevent live variables from unnecessarily replaces old ones stored in memory
  - Synchronization
    - Compile-time analysis and warn for race condition

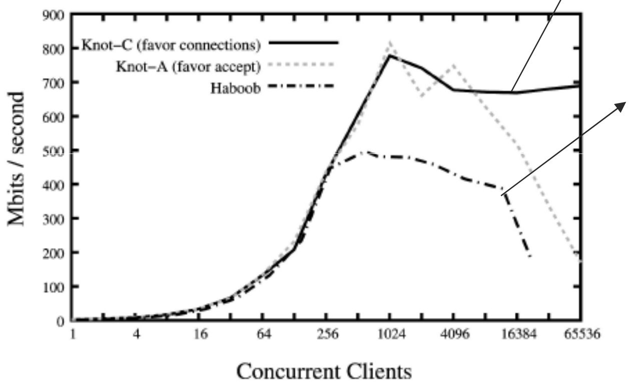
Evaluation for Highly Concurrent Thread Systems

- Benchmarking: Knot vs Haboob

Knot	Haboob
Thread-based web server	Event-based web server based on SEDA
Asynchronous I/O using UNIX poll() or sys_epoll()	Asynchronous I/O using Java NBIO
Thread pool for blocking I/O operation	Thread pool for event handling

Evaluation for Highly Concurrent Thread Systems

- Graphics showing performance



Steady on 700 Mbit/s bw

Degrading

- Favor connections -> favor processing of active connections over accepting new one
- Favor accept -> the reverse way

Personal Opinion

- The implication of this finding:
  - Compiler modification to support highly concurrent thread system -> modification is still not available
  - This won’t affect the business logic of higher layer -> infrastructure limitation not logic limitation
    - Provide options to use scalable threads or events via libraries when they are available