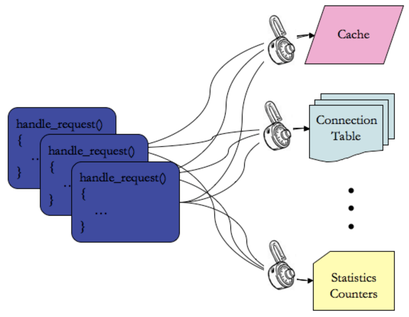
To be presented at 4:30 pm at the 2nd Workshop on Hot Topics in System Dependability (HotDep)

Safe at Any Speed: Fast, Safe Parallelism in Servers

John Jannotti and Kiran Pamnany
Department of Computer Science, Brown University
{jj,kiran}@cs.brown.edu

Multithreaded Servers

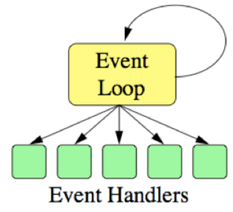
- Servers must take advantage of concurrency to handle their client loads
- Usual approach is multithreading
- Performance at the cost of correctness



- Thread accesses to *every* shared resource must be properly synchronized
- Miss one? Non-deterministic, hard-to-find "heisenbug"
- Locking is dangerous too--deadlock, livelock, priority inversion, convoying, starvation, etc.

Event-driven Servers

- Program registers interest in events (callbacks)
- Event loop waits for events; invokes handlers
- State stored in "context" which is passed as an argument when a handler is invoked



- No synchronization required
- Handlers are atomic blocks
- Single threaded
- Must use asynchronous calls; blocking stops progress
- Difficult to exploit multiprocessors

Approach

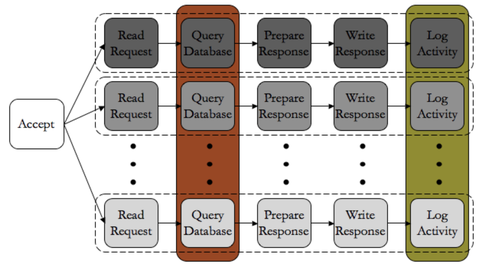
- Add concurrency without requiring synchronization
- Run event handlers in parallel when safe to do so

Static Program Analysis:

- Conservatively determine whether handlers share data unsafely
- Generate constraints on concurrent execution of handlers
- Provide detailed feedback--why do handlers conflict?

Runtime System:

- Run handlers concurrently subject to the constraints generated by the analysis



- Programmer removes constraints to increase performance

Philosophy

The Wrong Way:

- Start with concurrent, incorrect application
- Apply development effort until *all* races are fixed
- Incremental gains in correctness
- Miss something? Unsafe parallelism; incorrect program

The Right Way:

- Parallel applications should be *safe by default*
- Start with serial, correct application
- Apply development effort to add concurrency
- Incremental gains in performance
- Miss something? Loss of parallelism; performance problem
- Maintain correctness throughout

Future Work

- Program analyzer and runtime system in active development (using CIL and libevent)
- Evaluation (on thttpd)
- Beyond event-driven programs--multithreading

1. Static Program Analysis

- Conservative: may = will
- Enables default safety

```
handle_send(..., Context ctxt, ...)          handle_read(..., Context ctxt, ...)
{                                            {
    ...                                          ...
    glob_ctr++;          <-------------->        if (glob_ctr > 0)
    ...                                              ...
    ctxt->state = DONE;  <-------------->        ctxt->state = SEND;
    ...                                          ...
}                                            }
```

Conflict on global:

- handle_send() reads and writes a global; handle_read() accesses the same global
- Unsafe to run concurrently under any circumstances

Conflict through context:

- Both handlers update the same element
- Unsafe to run concurrently only if contexts are the same

2. Constraints

Handler	A()	B()	C()	D()	...
A()	11				
B()	10	11			
C()	00	00	01		
D()	10	01	00	11	
...					

- Two bits per cell
- Bit 0 is on if conflict on global
- Bit 1 is on if conflict through context
- A() conflicts with B() through the context; they can run concurrently if their contexts are known to be different
- C() conflicts only with itself on a global; it can run concurrently with every other handler
- B() conflicts with D on a global; they can never run concurrently

3. Hue/Color Scheduling

- Conservative approximation of constraints
- Queue per hue and queue per color
- Hue queues feed color queues
- Only one pending handler invocation of a given hue in the color queues at any time