Why Discuss Web Server in OS Class?

• Paper discusses design of well-conditioned Web servers
• Thread-based and event-driven concurrency are central to OS design
• Task scheduling and resource management issues also very important
Well-conditioned service (is the goal)

• Should behave in pipeline fashion:
  – If underutilized
    • Latency (s) = N x max stage delay
    • Throughput (requests/s) proportional to load
  – At saturation and beyond
    • Latency proportional to queue delay
    • Throughput = 1 / max stage delay
    • Graceful degradation

• Not how Web typically performs during “Slashdot effect”
Thread-based concurrency

- One thread per request
- Offers simple, supported programming model
- I/O concurrency handled by OS scheduling
- Thread state captures FSM state
- Synchronization required for shared resource access
- Cache/TLB misses, thread scheduling, lock contention overheads
Threaded server throughput versus load

- Latency is unbounded as number of threads increases
- Throughput decreases
- Thrashing – more cycles spent on overhead than real work
- Hard to decipher performance bottlenecks
Bounded thread pool

- Limit the number of threads to prevent thrashing
- Queue incoming requests or reject outright
- Difficult to provide optimal performance across differentiated services
- Inflexible design during peak usage
- Still difficult to profile and tune
Event-driven concurrency

- Each FSM structured as network of event handlers and represents a single flow of execution in the system
- Single thread per FSM, typically one FSM per CPU, number of FSM’s is small
- App must schedule event execution and balance fairness against response time
- App must maintain FSM state across I/O access
- I/O must be non-blocking
- Modularity difficult to achieve and maintain
- A poorly designed stage can kill app performance
Event-driven server throughput versus load

- Avoids performance degradation of thread-driven approach
- Throughput is constant
- Latency is linear
Structured event queue overview

• Partition the application into discrete stages
• Then add event queue before each stage
• Modularizes design
• One stage may enqueue events onto another stage’s input queue
• Each stage may have a local thread pool
A SEDA stage

- Stage consists of:
  - Event queue (likely finite size)
  - Thread pool (small)
  - Event handler (application specific)
  - Controller (local dequeueing and thread allocation)
A SEDA application

- SEDA application is composed of network of SEDA stages
- Event handler may enqueue event in another stage’s queue
- Each stage controller may
  - Exert backpressure (block on full queue)
  - Event shed (drop on full queue)
  - Degrade service (in application specific manner)
  - Or some other action
- Queues decouple stages, providing
  - Modularity
  - Stage-level load management
  - Profile analysis/monitoring
  - With increased latency
Controllers dynamically tune resource usage to meet performance targets

- May use both local stage and global state
- Paper introduces implementations of two controllers (others are possible)
  - Thread pool – create/delete threads as load requires
  - Batching – vary number of events processed per stage invocation
Asynchronous I/O

- SEDA provides I/O stages:
  - Asynchronous socket I/O
    - Uses non-blocking I/O provided by OS
  - Asynchronous file I/O
    - Uses blocking I/O with a thread pool
Asynchronous socket I/O performance

- SEDA implementation provides fairly constant I/O bandwidth
- Thread pool implementation exhibits typical thread thrashing
Performance comparison

- SEDA Haboob vs Apache & Flash Web servers
- Haboob is complex, 10 stage design in Java
- Apache uses bounded process pools in C
  - One process per connection, 150 max
- Flash uses event-driven design in C
- Note: authors claim creation of “Haboob” was greatly simplified due to modularity of SEDA architecture
I got a Haboob.

**ha·boob**

/ˈheɪbub/ [huh-boob]

—noun a thick dust storm or sandstorm that blows in the deserts of North Africa and Arabia or on the plains of India.

From www.dictionary.com
• Apache fairness declines quickly past 64 clients
• Throughput constant at high loads for all servers, Haboob is best
• Apache and Flash exhibit huge variation in response times (long tails)
• Haboob provides low variation in response times at cost of longer average response times
Performance comparison (cont.)

- Apache, Haboob w/o controller process all requests, buggy Flash drops ¾
- Haboob response time with controller better behaved
  - Controller drops requests with error notification under heavy load
  - Here 98% of requests are shed by the controller at bottleneck
  - Still not able to offer guarantee of service better than target (22 vs. 5)
Conclusion

- SEDA provides a viable and modularized model for Web service design
- SEDA represents a middle ground between thread- and event-based Web services
- SEDA offers robust performance under heavy load, optimizing fairness over quick response
- SEDA allows novel dynamic control mechanisms to be elegantly incorporated