SEDA: An Architecture for Well-Conditioned Scalable Internet Services
Matt Welsh, David Culler, and Eric Brewer
SOSP 2001

Presented by: Tom Harke
For: CS 533 - Operating Systems
8 April 2009
Outline

1. Introduction
   - Motivation
   - Example
   - Threads vs. Events

2. SEDA

3. Validation

4. Conclusion
Acknowledgements

Reused material by

Dan Coates   CS 533, Winter 2009
Jarett Creason CS 533, Winter 2008
SEDA website www.eecs.harvard.edu/~mdw/proj/seda
Motivation

Desire an architecture for web services

- Good performance
- Handle extreme fluctuations
  - Slashdot effect
- Versatile
- Easy for application writers to use
  - Separates concurrency details from application logic
Running Example - Simple HTTP server

- At high level: Finite State Machine
- Can implement with threads, events, or SEDA
The next few slides examine the 4 entries of this table

<table>
<thead>
<tr>
<th></th>
<th>Threads</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>straightforward ...</td>
<td>ad hoc ...</td>
</tr>
<tr>
<td>Performance</td>
<td>degrades ...</td>
<td>scales ...</td>
</tr>
</tbody>
</table>
Given many processors working on many tasks... 

- What’s the best case when there’s too much work? 
  - What’s the ideal throughput? 
  - What’s the ideal latency? 

- Well-conditioned 
  - Behaves like simple pipeline 
  - Degradation only due to load saturation
Threads - Design

Good

- Thread-per-request
- Straightforward to program
- I/O implicitly handled

Bad

- Overheads: scheduling, cache misses, lock contention
- Can bound thread pool – rejection, unfairness
- Hard to observe bottlenecks
- Hard to tune
Threads - Performance

- Far from ideal!
- Throughput decreases
- Latency greater than linear
Events - Performance

- This performance is ideal, what’s to improve?
Events - Design

Good
- Well-conditioned performance

Bad
- Ad-hoc, complex
- Not modular
- Scheduling policy depends on application
- Needs non-blocking I/O
Outline

1. Introduction

2. SEDA
   - Overview
   - Example
   - Components
   - Asynchronous I/O

3. Validation

4. Conclusion
**Staged Event-Driven Architecture**

- Hybrid Architecture
  - Sibling to Thread-based and Event-based
  - Closer to Event-based
  - Client application sees events
  - Under the hood it uses threads

- Better than threads – performance
  - Closer to ideal throughput, latency
  - (Can be) fair when overburdened

- Better than events – ease of use
  - Modular
  - Can observe & tune performance
  - Concurrency separate from application logic
Simple HTTP Server in SEDA

Stage has
- Event handler
- Incoming event queues
- Thread pool

Stage boundary
- Where a task may be paused
- Controllers watch Queues
- Other opportunities for profiling, debugging
Application programmer...

- Writes app in style of
  - Finite State Machine +
  - Event handlers
- Selects controllers
Example Controllers

Dynamic tuning

- Resizing thread pools
- Changing batching factor
- Reordering within batch
- Throttling rate of packets
- Load shedding

How? Queues are observable
Asynchronous Socket I/O

- Event model requires non-blocking I/O
- Underlying Sockets are already asynchronous – just need simple wrapper
Asynchronous Sockets Layer Performance

(Can’t run beyond 400 connections)

Bandwidth, Mbit/sec

Number of connections

SEDA asynchronous socket layer
Thread-based asynchronous socket layer
Underlying file I/O blocks

- add a stage to handle file I/O
- thread(s) may block, but event queue decouples this stage from original caller
- careful to preserve sequential modifications
- Bounded thread pool – same limitations as thread-based sockets
Outline

1. Introduction
2. SEDA
3. Validation
   - Haboob
4. Conclusion
- **SEDA**
  - the design idea

- **Sandstorm**
  - an implementation of SEDA
  - proof of concept, in Java

- **Two applications using Sandstorm**
  - Haboob (A HTTP server)
  - A Gnutella peer-to-peer router
Modular: Able to change handler for page caching, filesystem interface without changing other parts
Haboob Performance – Moderate Load

- Haboob vs Apache (Thread-based, 150 threads) vs Flash (Event-based)
- Apache maintains throughput by refusing connections
- Goal: Short tails on cumulative distribution
- Low variance better than low average
Haboob Performance – Heavy Load

An example strategy:
- Queue full $\Rightarrow$ drop request
- Error message to client
- Adjust queue size, based on response time

Other strategies:
- Backpressure: block when full

Graph showing probability of response time vs. response time (msec) for Haboob with control, Haboob no control, Apache, and Flash.
Step towards best features of Threads and Events

- Performance of events
- Ease of use of threads (?)

Haboob is promising

- Edges out Flash, Apace in reducing latency outliers

Seems to be:

- Framework to enforce programming discipline, +
- I/O library +
- Controller library