SEDA: An Architecture for Well-Conditioned, Scalable Internet Services

Matt Welsh, David Culler, and Eric Brewer
Computer Science Division
University of California, Berkeley

Symposium on Operating Systems Principles (SOSP), October 2001

http://www.eecs.harvard.edu/~mdw/proj/seda/
Motivation

- Prototypical application: Web server
- Must handle “Slashdot effect” gracefully
- *Well-conditioned* service: simple pipeline with graceful degradation beyond load saturation
Anatomy of a simple HTTP server

- Finite state machine (FSM)
- Can be implemented using threads or events (or SEDA)

Source: http://www.eecs.harvard.edu/~mdw/proj/seda/
Thread-based concurrency

- Create thread per task
- Straightforward to program
- I/O concurrency implicitly handled – thread state captures FSM state during blocking
Threaded server degradation

- Doesn't scale well due to context overhead at high load
- Throughput decreases
- Latency curve worse than linear
- Thrashing – more time spent on system overhead than real work
Bounded Thread Pool

- Limit the number of threads
- Make requests wait for a thread to become available or reject additional connections
- Common solution (Apache, IIS, Netscape Server, etc.)
- BUT:
  - How to determine the number of threads to use?
  - Unfair at load saturation: long waiting times
  - Difficult to profile and tune
Event-based model

- FSM structured as sequence of event handlers
- Single thread run event handlers
- I/O must be non-blocking
- FSM state must be explicitly recorded across I/O events
Event-based performance

- Throughput is constant, latency is linear with number of tasks
- Why better performance?
  - No virtualization of resource management
  - Lighter-weight contexts
  - Application has control of scheduling instead of OS
Disadvantages of events

- More difficult to program:
  - Need to maintain state throughout FSM
  - Need to manage continuations across I/O
  - Application responsible for explicit control of scheduling
- No principled framework, so solutions are typically ad-hoc, lacking modularity
- SEDA claims to be the first to offer general-purpose architecture
Staged Event-Drive Architecture (SEDA)

- Decompose applications into independent **stages**
- Separate stages by **queues**
- Each stage has its own thread pool for concurrent execution
- Queues provide clean boundary for modularity, security and profile analysis/introspection (at cost of increased latency)

Source: [http://www.eecs.harvard.edu/~mdw/proj/seda/](http://www.eecs.harvard.edu/~mdw/proj/seda/)
Staged Event-Drive Architecture (SEDA)

- **Finite** event queues can provide:
  - **backpressure**: blocking on a full queue
  - **load shedding**: dropping events
  - **errors** to user, etc.

Source: [http://www.eecs.harvard.edu/~mdw/proj/seda/](http://www.eecs.harvard.edu/~mdw/proj/seda/)
A SEDA Stage

- Modular application component
- Event queue and thread pool managed by SEDA machinery,
- Parameterized by application
- Controller dynamically adjusts resource allocation (optionally)
- Application-supplied event handler
Dynamic Resource Controllers

- Automatic, adaptive performance tuning
- No programmer intervention necessary

Dynamically adjust number of threads allocated to stage based on actual measured usage

Process variable number of requests during each time-step, for cache optimization and task aggregation
Asynchronous I/O

- Usage of event queues requires asynchronous I/O
- When OS provides asynchronous I/O, SEDA provides a natural wrapper layer for applications to use (example: socket I/O)
- If the OS only provides blocking file I/O, need to explicitly create a stage with a bounded thread pool
Asynchronous I/O Performance

- \textit{asyncSocket} layer
- Uses non-blocking /dev/poll
- Performance compared to blocking sockets and thread pool
Results: “Haboob” SEDA Web Server

- More complicated Web server, broken into stages
- Compared to **Apache** (150 static processes) and **Flash** (event-based)
Results: “Haboob” SEDA Web Server

- Note decline in fairness for Apache
- While having high frequency of low response times, note heavy tail (with long response times) for Apache and Flash

<table>
<thead>
<tr>
<th>Server</th>
<th>Throughput</th>
<th>RT mean</th>
<th>RT max</th>
<th>Fairness</th>
<th>Throughput</th>
<th>RT mean</th>
<th>RT max</th>
<th>Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>173.36 Mbps</td>
<td>143.91 ms</td>
<td>27953 ms</td>
<td>0.98</td>
<td>173.09 Mbps</td>
<td>475.47 ms</td>
<td>93691 ms</td>
<td>0.80</td>
</tr>
<tr>
<td>Flash</td>
<td>180.83 Mbps</td>
<td>141.39 ms</td>
<td>10803 ms</td>
<td>0.99</td>
<td>172.65 Mbps</td>
<td>665.32 ms</td>
<td>37388 ms</td>
<td>0.99</td>
</tr>
<tr>
<td>Haboob</td>
<td>208.09 Mbps</td>
<td>112.44 ms</td>
<td>1220 ms</td>
<td>0.99</td>
<td>201.42 Mbps</td>
<td>547.23 ms</td>
<td>3886 ms</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Results: Resource Throttling

- 1024 clients repeatedly request dynamic pages with I/O and computation
- Apache and vanilla Haboob process all requests, Flash quietly drops
- Haboob controller drops (with error) if average response time > 5 sec
Summary

- SEDA provides a principled framework for implementing highly concurrent event-based Web services
- Based on the published results, it has robust performance, especially at saturation
- Provides opportunities for both manual and automatic tuning