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/

for:

CS533 - Concepts of Operating Systems
Portland State University,
Professor Jonathan Walpole

presented by: Dan Coates

January 12th, 2009
Motivation

- Prototypical application: Web server
- Must handle “Slashdot effect” gracefully
- *Well-conditioned* service: simple pipeline without degradation beyond load saturation
- Unlike traditional OS load balancing concerns

(source: Wikipedia)
SEDA

• From Matt Welsh's PhD work at UC Berkeley (2000-2002)

• Architecture for highly concurrent server applications

• Hybrid design combines elements of threaded and event-based paradigms

• Programming framework attempts to separate application logic from OS concurrency details while providing performance control and profiling in a principled way
Outline

● Thread vs. event-based approaches (to Web services)
● SEDA model
● SEDA specifics: I/O and dynamic resource allocation
● Results
● Summary
● Criticisms/Questions
Anatomy of a simple HTTP server

- Finite state machine (FSM)
- Can be implemented using threads or events (or SEDA)
- Need to consider blocking in each node
Thread-based concurrency

- Create thread per task
- Straightforward to program
- I/O concurrency implicitly handled
Threaded server degradation

- Doesn't scale well due to context overhead at high load
- Throughput decreases
- Latency curve worse than linear
Bounded Thread Pool

- Limit the number of requests, reject additional connections
- Common solution (Apache, IIS, Netscape Server, etc.)
- BUT:
  - How to determine number of threads to use?
  - Is unfair at load saturation: long waiting times
  - Difficult to profile and tune
Event-based model

- Single thread handles requests, no concurrency issues
- Event handlers should be short-lived
- I/O must be non-blocking
Event-based performance

- Better performance: program has control of scheduling instead of OS, so lighter-weight contexts
- Constant throughput up to 1M tasks
- Latency is linear with number of tasks
Disadvantages of events

- More difficult to program:
  - Need to maintain state throughout FSM
  - Application responsible for explicit control of scheduling
- No principled framework, so solutions are typically ad-hoc, lacking modularity
- “Structured event queues” attempt to provide modular designs, although SEDA was first to offer general-purpose architecture.
Staged Event-Drive Architecture (SEDA)

- Decompose applications into independent **stages** separated by queues
- Each stage has its own thread pool for concurrent execution
- Separate 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/
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/
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

(a) Thread pool controller
Dynamically adjust number of threads allocated to stage based on actual measured usage

(b) Batching controller
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 I/O, need to explicitly create a stage with a bounded thread pool (example: file I/O)
Asynchronous I/O Performance

- asyncSocket layer
- Uses non-blocking /dev/poll
- Performance compared to blocking sockets and thread pool
Results: Sandstorm

- Sandstorm is a Java proof-of-concept platform to provide web services
- Authors have implemented a moderately simple Web server, and a peer-to-peer file sharing system

```
<stages>  # Specify stages
  <HttpRecv>  # Packet receiver
  </initargs>
</HttpRecv>

  <HttpSend>  # Packet sender
    class sedia.apps.Habooob.http.HttpSend
  </initargs>
</HttpSend>

  <BottleneckStage>  # Artificial bottleneck
    class sedia.apps.Habooob.bottleneck.Bottleneck
  </initargs>
</BottleneckStage>

<ThreadPool>  # Initialthreads 1/sizecontroller 1
  <initargs>
    threshold 1  # Set lower threshold for adding threads
  </initargs>
</ThreadPool>

</DynamicHttp>
```

```
public void handleEvent(queueElement item) {
    if (DEBUG) System.err.println("HttpRecv: GOT DEL: " + item);

    if (item instanceof httpConnection) {
        HabooobStats.numConnectionsEstablished++;
        numConns++;
        if (VERBOS) System.err.println("HttpRecv: Got connection " +
                               (HabooobStats.numConnectionsEstablished - HabooobStats.numConnectionsClosed));

        if (((maxConns == -1) || (numConns == maxConns)) {
            System.err.println("Suspending accept() after "+numConns+" connections");
            server.suspendAccept();
        }
    } else if (item instanceof httpRequest) {
        if (DEBUG) System.err.println("HttpRequest: Got request " + item);

        httpRequest req = (httpRequest)item;

        // Record time for controller
        req timestamp = System.currentTimeMillis();

        // Check for special URL
    }
```
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
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 Web services
- The hybrid approach attempts to get best of both worlds: performance, with flexibility and ease of programming
- Based on the published results, it has robust performance, especially at saturation
- Ample opportunities for both manual and automatic tuning
Criticisms/Questions

● While much mention was made of the need for supporting “dynamically changing services,” the Web server performance methodology used:
  - only static Web pages
  - loops with read, 20 ms sleep. More thorough testing employs randomization to more closely approximate real-world performance

● Does the effort in learning SEDA pay off? Only a handful of applications have employed the system in the last 7 years.

● I'm curious how much OS specifics (Linux 2.2.14, /dev/poll, JVM, etc.) impacted the results.
Acknowledgments

- I made use of material from the SEDA website, http://www.eecs.harvard.edu/~mdw/proj/seda/.
- Many of the screenshots from the SEDA paper were reused from previous class presentations, especially Jarett Creason's presentation from January 2008.

Thank you for your attention!