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

By: Matt Welsh, David Culler, and Eric Brewer

Presenter: Hong Quach
Portland State University
CS 533 - Fall 2013
Overview

- Introduction
- Background and Related Work
  - Thread-based concurrency
  - Bounded thread pools
  - Event-driven concurrency
  - Structured event queues
- The Staged Event-Driven Architecture
  - Main building blocks -- stages
  - Network of stages
  - Dynamic resource controllers
- Applications and Evaluation
Problems with Internet Applications

1. Wide variation in loads:
   a. Certain time of day
   b. Sudden popularity of the site
   c. Replication solutions become not feasible

2. Generality of services:
   a. Require more computational power
   b. Logic tends to change rapidly
   c. Host on general-purpose facilities

3. Limited resource management:
   a. A need for massive concurrency
   b. A need for extensive control for load balancing
Introduction

• A high performance internet application to provide services that must be responsive, robust, and always available.

• SEDA = Staged Event-Driven Architecture
  ○ An architecture for highly concurrent server applications
  ○ Combines thread-based concurrency model and event-based model
Thread-based concurrency

- Model: Thread-per-request -- spawn a new thread to handle each new request (from start to finish, including I/O)
- Used in: RPC, Java-RMI, and DCOM
Super Store Analogy

1 store = 1 system
1 worker = 1 thread
1 service* to 1 customer = 1 task

Thread-based concurrency

● Hire one worker to service each customer

What are the Pros and Cons?

*Checkout, help find an item, answer a question, etc...
Thread-based concurrency

Pros:
- One thread per request
- Relatively easy to program
  - Follow the multi-thread programming model
  - Protect critical section

Cons:
- Overheads associate with each threads
- Massive concurrent threads could lead to system crash
Thread-based concurrency

Threaded server throughput degradation

- 1 thread per request
Bounded thread pools

- Same as threaded-base concurrency except the number of threads is bounded to a limit
- Used by: Apache, IIS, Netscape ES, BEA Weblogic, and IBM WebSphere
- An obvious fix to the thread-based concurrency problem
Super Store Analogy

1 store = 1 system
1 worker = 1 thread
1 service to 1 customer = 1 task

Bounded thread pools

- Hire one worker to service each customer
- Limit the number of workers

What are the Pros and Cons?
Bounded thread pools

Pros:
- One thread per request
- Relatively easy to program
  - Follow the multi-thread programming model
  - Protect critical section

Cons:
- Introduce unfairness to client requests
  - All requests are not created equally
  - Stop accepting requests when server saturated
- Hard to identify performance bottlenecks
Event-driven concurrency

- Process each task as triggered by event
- Sources of event: disk I/O, network I/O, application events, and timer.
Super Store Analogy

1 store = 1 system
1 worker = 1 thread
1 service to 1 customer = 1 task

Event-driven concurrency

- **Hire one worker to service each customer**
- **Limit the number of workers**
- **Worker only provides service when asked**

What are the Pros and Cons?
Event-driven concurrency

Pros:
● Tends to be robust to load
● Maintain high throughput
● More control over the scheduling

Cons:
● Manage the scheduling and ordering of events
  ○ When and in what order to process incoming events
  ○ Scheduling algorithm is often tailored to specific application, potential redesign for new functionality
  ○ Modularity is hard to achieve
Event-driven concurrency

Event-driven server throughput

- 1 Thread with increasing tasks
Structured event queues

- Variants of the Event-Driven Concurrency model by partitioning the main event queue into multiple sub-event queues
- Used in: Click modular package router, Gribble’s DDS layer, Work Crews, TSS/360 queue scanner, and StagedServer system
- Each variant carefully structures the event queues to achieve its goal
Restate the pros of different models

Thread-based concurrency model:
● One thread per request
● Relatively easy to program
  ○ Follow the multi-thread programming model
  ○ Protect critical section

Event-driven concurrency model:
● Tends to be robust to load
● Maintain high throughput
● More control over the scheduling
The Staged Event-Driven Architecture

Goals:

- Support **massive** concurrency
- Simplify the construction of well-conditioned services
- Enable introspection
- Support self-tuning resource management

SEDA’s fundamental building block -- stage

- an event handler
- an incoming event queue
- a thread pool
- a controller (the secret sauce)
Super Store Analogy

1 store = 1 system
1 worker = 1 thread
1 service to 1 customer = 1 task

SEDA - staged event-driven architecture

- Hire one worker to service each customer
- Limit the number of workers
- Worker only provides service when asked
- Partition the workers into separate teams and each team will also get a team leader

What are the Pros and Cons?
Application as a network of stages

- Stages connected by even queues
- Event handler enqueues events onto another stage’s event queue
- Using event queue as an interface between stage help set control boundary
Application as a network of stages

- Stages connected by even queues
- Event handler enqueues events onto another stage’s event queue
- Using event queue as an interface between stage help set control boundary

- Should modules “to be, or not to be” treated as stages?
Application as a network of stages

- Stages connected by even queues
- Event handler enqueues events onto another stage’s event queue
- Using event queue as an interface between stage help set control boundary

- Should modules “to be, or not to be” treated as stages?
Dynamic resource controllers

- No need to do performance tuning
- SEDA auto adjust the processing power of each stage based on the performance and demand
- Possible to implement more complex control

(a) Thread pool controller  (b) Batching controller
SEDA Thread pool controller
SEDA Adaptive load shedding

- Add a new stage to monitor the average response time of request passing through the bottleneck stage.
- Control the stage queue operation when the response time exceeds a threshold.
- Handle the “failed” enqueue operation (reject or redirect).
- Flash has a bug that silently rejects connection.
Asynchronous I/O Primitives

High concurrency requires efficient robust I/O interface:

- **Asynchronous socket I/O**
  - Process each request by making non-blocking calls to the corresponding socket stages: `readStage`, `writeStage`, and `listenStage`.

- **Asynchronous file I/O**
  - Process each request by performing the corresponding I/O (blocking)
  - One thread to operate on a particular file at a time
Haboob: A high performance HTTP server

(a) Throughput vs. number of clients

(b) Cumulative distribution of response time for 1024 clients

<table>
<thead>
<tr>
<th></th>
<th>256 clients</th>
<th></th>
<th>1024 clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Throughput</td>
<td>RT mean</td>
<td>RT max</td>
</tr>
<tr>
<td>Apache</td>
<td>173.36 Mbps</td>
<td>143.91 ms</td>
<td>27953 ms</td>
</tr>
<tr>
<td>Flash</td>
<td>180.83 Mbps</td>
<td>141.39 ms</td>
<td>10803 ms</td>
</tr>
<tr>
<td>Haboob</td>
<td>208.09 Mbps</td>
<td>112.44 ms</td>
<td>1220 ms</td>
</tr>
</tbody>
</table>
The set of stages: *GnutellaServer*, *GnutellaRouter*, *GnutellaCatcher*, and asynchronous socket I/O layers.

From a 37-hour run, the router processed 24.8M packages, received 72,396 connections, and average of 12 simultaneous connection at any given time.
Gnutella package router latency

(a) Using single thread

(b) Using thread pool controller

(c) Queue length profile
Review SEDA’s Goals

✓ Support **massive** concurrency
Review SEDA’s Goals

✓ Support **massive** concurrency
✓ Simplify the construction of well-conditioned services
Review SEDA’s Goals

✓ Support **massive** concurrency
✓ Simplify the construction of well-conditioned services
✓ Enable introspection
Review SEDA’s Goals

- Support **massive** concurrency
- Simplify the construction of well-conditioned services
- Enable introspection
- Support self-tuning resource management
Discussion and Conclusion

- Massive concurrency is needed for high performance application as more connected computing device being added as time goes on.
- SEDA is one of the approach for design and implement high performance applications.
- The modularity of stages connected by queues introduce isolation that help in debug of application.
- Applications that can manage the resource usage would perform better by dynamically assign the resource to handle bottlenecks.
- “With great power come with great responsibility”
  - How to detect overload condition?
  - What to do to prevent overload?
Discussion and Conclusion

- Programming in the SEDA model is easier than multithreaded application design and traditional event-driven model
- Should operating system expose more control over resource management to the application?

Computer Layer

- Application - Makes customer happy
- Operating System - Interfaces between Apps and HW
- Hardware - flips the switch