SEDA: An Architecture for Well-Conditioned Scalable Internet Services

Matt Welsh, David Culler, and Eric Brewer

presented by Ahn, Ki Yung
Staged Event-Driven Architecture

- Designed for highly concurrent Internet services
- Applications are network of stages
- Stages are driven by event
- Stages connected by explicit event queues
Internet is a problem

- millions of users demanding access
- more complex and dynamic contents
- traditional OS design does not fit
  - multiprogramming for resource virtualization
- replication, clustering not always suffice
  - peak load may seldom occur
Well-Conditioned service

- Behaving like a simple pipeline
- Throughput increase proportional to the load, until it saturates (pipeline full)
- Graceful Degradation: when overloaded,
  - throughput does not degrade
  - linear response-time penalty equally
Figure 2: Threaded server throughput degradation: This benchmark mea-
Figure 4: Event-driven server throughput: This benchmark measures...
Concurrency Models

- **Thread-per-Request**
  - throughput degrades for large # of access

- **Bounded Thread Pools**
  - can avoid throughput degradation
  - response time may be extremely unfair

- **Event-Driven**
  - robust to load
  - event handler should not block
  - application should schedule & order events
Figure 1: **Threaded server design:** Each incoming request is dispatched to a

Figure 3: **Event-driven server design:** This figure shows the flow of events
SEDA goals

- massive concurrency
  - event-driven execution, asynchronous IO

- simplify construction
  - provide scheduling, resource management

- enable introspection on event queues
  - application can have control on events

- self-tuning resource management
  - thread pool controller, batching controller
Conclusion & Discussion

- SEDA is a combination of threaded model and event-driven model
- event-driven stages - modularity
- explicit event queues - control over events
- dynamic controllers
  - scheduling & resource management
- easier to build well-conditioned service
- SEDA can be new OS design model
  - more control over scheduling & resource
  - shared virtualized resource not necessary
Figure 5: **Staged event-driven (SEDA) HTTP server:** This is a structural representation of the SEDA-based Web server, described in detail in Section 5.1. The application is composed as a set of stages separated by queues. Edges represent the flow of events between stages. Each stage can be independently managed, and stages can be run in sequence or in parallel, or a combination of the two. The use of event queues allows each stage to be individually load-conditioned, for example, by thresholding its event queue. For simplicity, some event paths and stages have been elided from this figure.

Figure 6: **A SEDA Stage:** A stage consists of an incoming event queue, a thread pool, and an application-supplied event handler. The stage’s operation is managed by the controller, which adjusts resource allocations and scheduling dynamically.

Figure 7: **SEDA resource controllers:** Each stage has an associated controller that adjusts its resource allocation and behavior to keep the application within its operating regime. The thread pool controller adjusts the number of threads executing within the stage, and the batching controller adjusts the number of events processed by each iteration of the event handler.
SEDA seem familiar to what I’ve seen before

- Distributed Systems
  - Computers : stages
  - OS (TCP stack) : event queue & scheduler

SEDA is like a model implementation of distributed system in a single machine
  - That is ... MULTIPROGRAMMING with IPC?
Questions

- How did they implement the SEDA queue?
- How to arrange events and asynchronous IO?
- Is SEDA essentially different from multiprocess/thread programming with message passing OR distributed systems?