Architecture for Scalable Internet Services

: SEDA
Motivation

Internet Service

- Unpredictable scale: Slashdot Effect
- Huge variation in service load

[Image of Slashdot webpage]
Motivation

- Large Scale Means..
  - Complex Service: Static -> Dynamic
  - Rapidly Changing
  - Running in General purpose device

- It causes..
  - Millions of Access
  - Tremendous concurrent session
  - Higher Network & IO request
Motivation

- It is not easy on commodity OS..
  - Dedicated to virtual environment
  - Not aimed at mass concurrency
  - Not aimed at extensive resource control
  - context-switching, cache, TLB stall
Thread-based concurrency

- Thread per Request
- RMI, DCOM, almost every system
Research

- Thread-based concurrency
  - It works TRANSPARENTLY
  - cache, TLB miss, scheduling, lock

![Graph showing throughput and latency vs. number of threads]
Bounded thread pool
- Restrict thread pool
- If over boundary, no more connection
- Apache, IIS, WebLogic, WebSphere
- Throughput? OK.
- But UNFARE! - queue up and waiting - arbitrarily large waiting times
Research

- Event driven concurrency
Research

- Event driven concurrency
  - Scheduler responsible entire procedure
  - Every FSM is one thread
  - I/O should be non-blocked type
  - Need mechanism, ex) helper process
  - Should be very careful for scheduler and ordering / non-blocking mechanism
  - Flash, thttpd, Zeus, JAWS Web servers
Event driven concurrency
Staged Event-Driven Architecture

- Event Queue
- Event Handler
- Thread Pool
- Controller

Stage
Suggestion
**Suggestion**

- **Staged Event-Driven Architecture**

---

**Diagram:**

- **Socket listen**
  - accept connection
- **Socket read**
  - read packet
- **HttpParse**
  - parse packet
- **HTTP request**
  - check cache
  - cache hit
  - cache miss
  - CacheMiss
  - handle miss
  - file I/O
  - I/O request
  - file data
- **PageCache**
  - send response
  - packet
  - Socket write
  - write packet
Dynamic Resource Controller

(a) Thread pool controller
(b) Batching controller
Resource Controller

- Thread Pool Controller
Resource Controller

- Batch Controller

![Graph showing Controller input and output over time](image)
Asynchronous I/O Primitives

- Asynchronous Socket I/O

Figure 10: SEDA-based asynchronous sockets layer: *The Sandstorm sock-*
Asynchronous I/O Primitives

Asynchronous Socket I/O

Bandwidth, Mbit/sec

Number of connections

(Can’t run beyond 400 connections)

SEDA asynchronous socket layer
Thread-based asynchronous socket layer
Asynchronous I/O Primitives

- Adaptive load shedding

```plaintext
1. Haboob with control
2. Haboob, no control
3. Apache
4. Flash
```

Prob [response time <= x] vs. Response time, msec
Evaluation

- Haboob HTTP Server

(a) Throughput vs. number of clients
Evaluation

- Haboob HTTP Server

(b) Cumulative distribution of response time for 1024 clients
Yet, we need more smart dynamic resource controller

SEDA is easy programming model

SEDA supporting OS
- much more control over scheduling and resource
- Prohibit applications from sharing resource transparently
Discussion

February 19, 2007 - A Note on the status of SEDA

.... a number of recent research demonstrated that the SEDA prototype (in Java) performs poorly compared to threaded or event-based systems implemented in C. (Vivek Pai's group at Princeton and the Capriccio work from UC Berkeley.)

It is important to keep a few things in mind when interpreting them. First, the SEDA implementation in Java. More recent studies have varied the environment substantially.

...This likely needs to be tuned or redesigned to support high bandwidth networks and more recent Linux and JVM implementations...

Tim Brecht's group at Waterloo has undertaken a study of competing Web server architectures and has shown that a SEDA implementation in C++, appropriately tuned, performs comparably to alternatives, so I do not believe these performance issues are fundamental to the architecture.

The most fundamental aspect of the SEDA architecture is the programming model that supports stage-level backpressure and load management. Our goal was never to show that SEDA outperforms other server designs, but rather that acceptable performance can be achieved while providing a disciplined approach to overload management.