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

Matt Welsh, David Culler, and Eric Brewer

Computer Science Division
University of California, Berkeley

Operating Systems Principles (SOSP-18), Chateau Lake Louise, Canada, October 21-24, 2001.

## Motivation

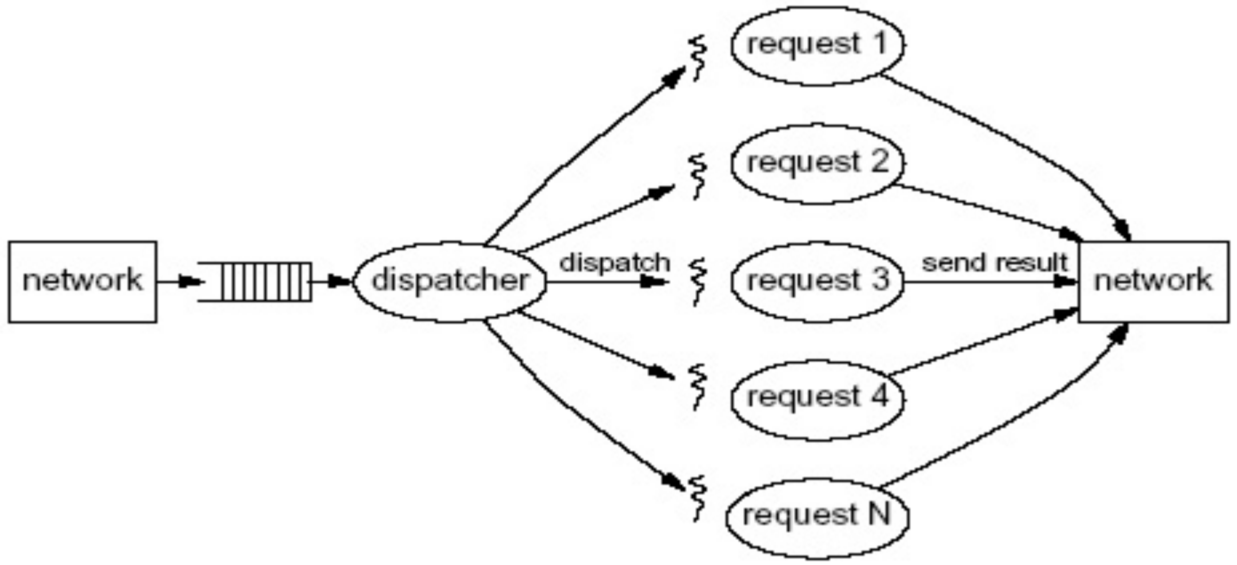
- Millions of Internet users
- Demand for Internet services grows
- High variations in service load. Load spikes are expected
- Web services getting more complex
- Not static content anymore, dynamic content that require extensive computation and I/O

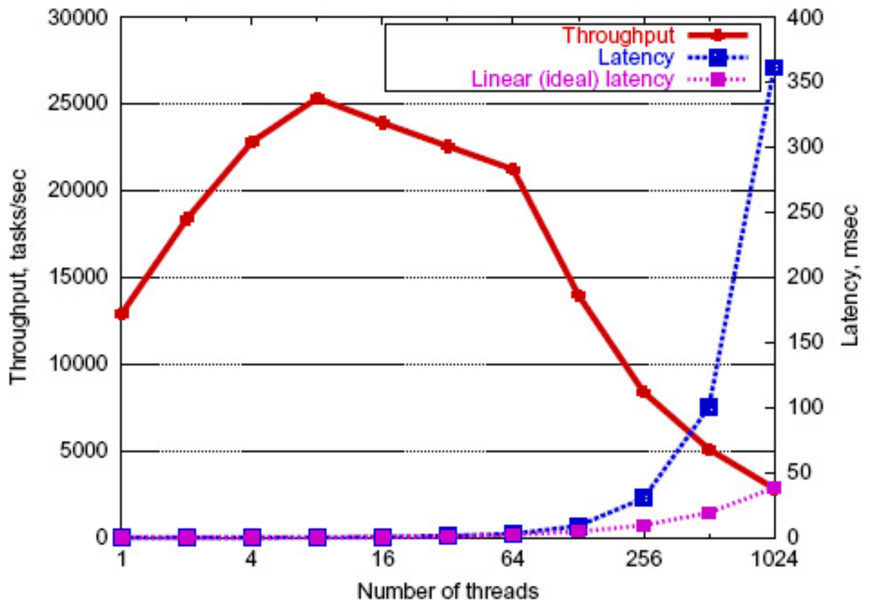
## Problem statement

- Services that support millions of users (massive concurrency)
  - Responsive
  - Robust
  - Highly available

- Well conditioned service :
  - Request rate scales with the response rate
  - Excessive demand does not degrade throughput and all clients experience an equal response time penalty linear to the length of the queue (graceful degradation).
  - A notion of fairness

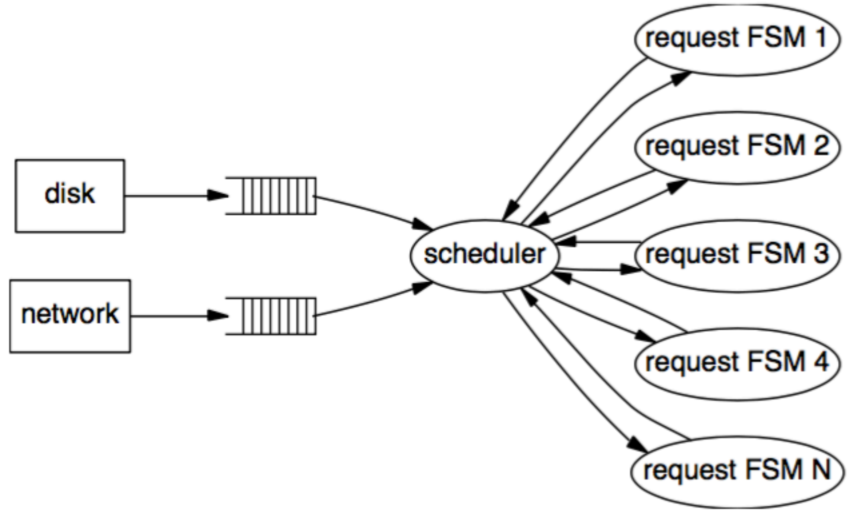
# Thread-based concurrency

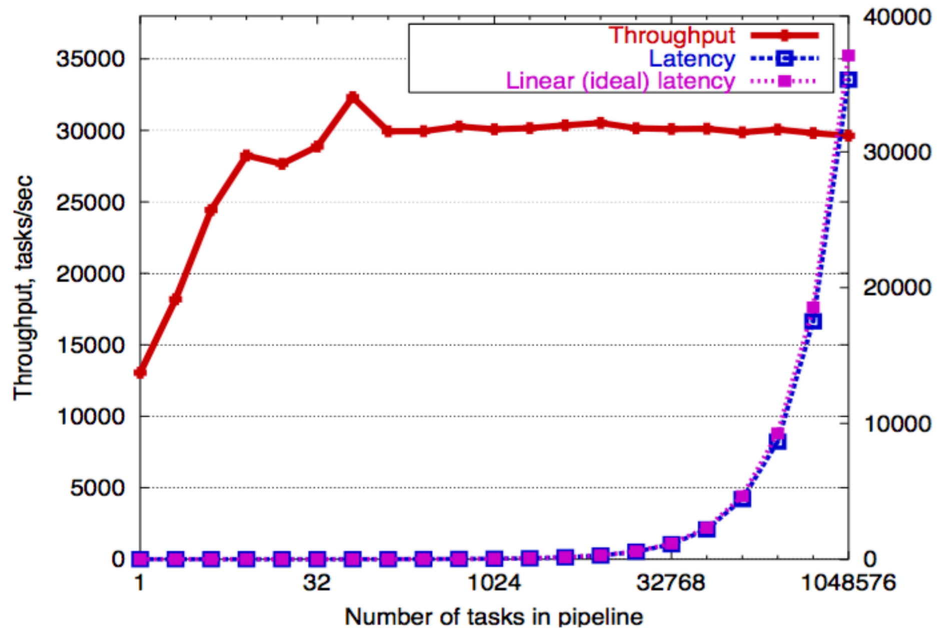




- Contention for resources and context switches cause high overhead
- High number of threads degrades the throughput and response time is greatly increased
- Partial Solution: Bound number of threads
  - Throughput maintained
  - But what about max response time? Some clients experience long waiting times
- Overcommitting resources
- Transparent resource virtualization prevents application from adapting to load changes and spotting bottlenecks

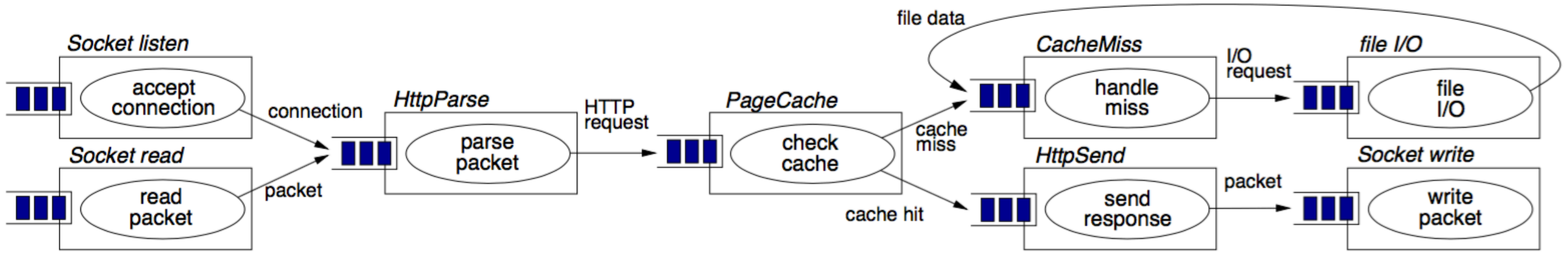
# Event-driven concurrency





- Efficient and scalable concurrency
- But difficult to engineer and tune
    - How order the processing of events. Scheduling challenges
    - Difficulty to follow the flow of events
    - Little support from OS

# Staged Event-Driven Architecture (SEDA)

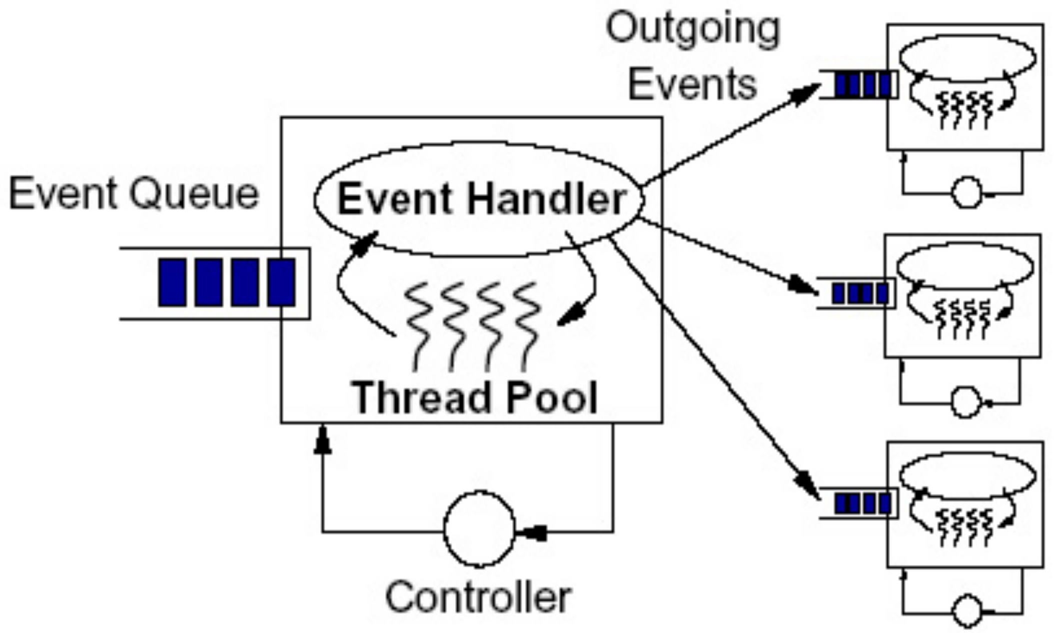


- Hybrid approach
  - Thread-based concurrency models for ease of programming
  - Event-based models for extensive concurrency

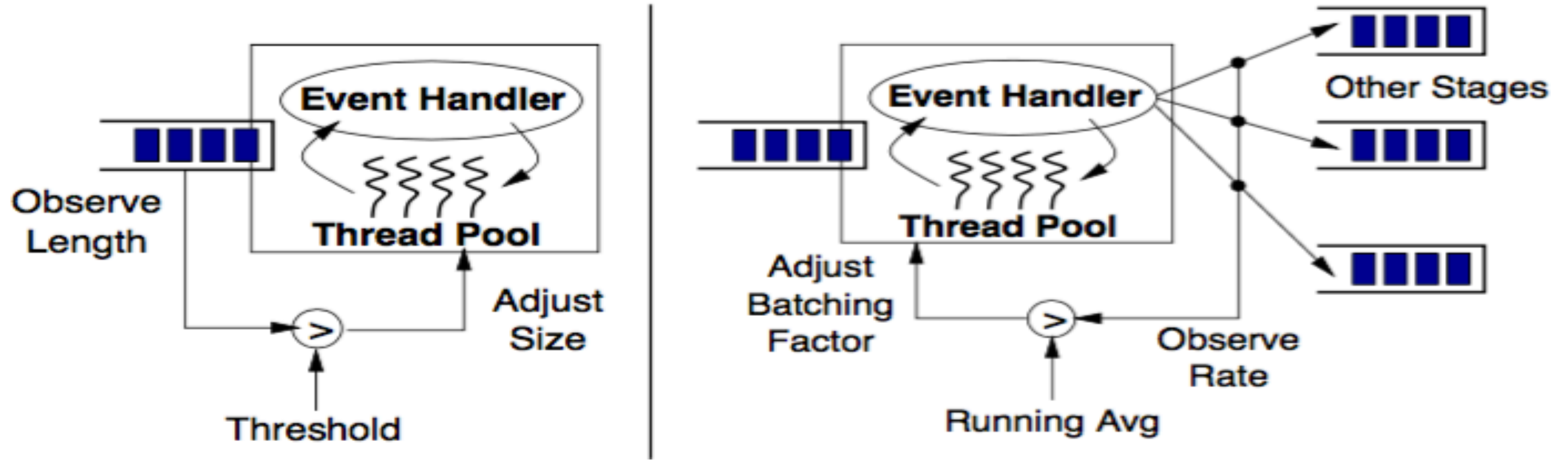
- Main Idea: Decompose service into stages separated by queues
  - Each stage performs a subset of request processing

# SEDA - Stage

- Event queues can pose various control policies
- Modularity. Each stage implemented and managed independently
- Explicit event delivery facilitates tracing flow of events and thus spotting bottlenecks and debugging



## Controllers



- Thread pool controller : ideal degree of concurrency for a stage
  - Adjust number of threads by observing the incoming queue length
  - Idle threads are removed

- Batching controller : aims at low response time and high throughput
  - Batching factor: number of events consumed at each iteration of the event handler
    - Large batching factor : more locality, higher throughput
    - Small batching factor : lower response time

# SEDA Prototype: Sandstorm

- Implemented in Java
  - Java provides software engineering benefits
    - Built-in threading, automatic memory management
- APIs are provided for naming, creating and destroying stages, performing queue operations, controlling queue thresholds and profiling and debugging
- Asynchronous I/O primitives are implemented using existing OS primitives.
  - The sockets interface consists of three stages: read, write and listen
  - Asynchronous I/O file operations.

## Evaluation

Evaluated Haboob a high-performance SEDA-based HTTP server

- Used the static file load from SpecWEB99 benchmark, a realistic, industry-standard benchmark
- 1 to 1024 clients making repeated requests
- Files sizes range from 102 to 921600 Bytes
- Total file set size is 3.31 GB
- Memory Cache of 200Mb
- Server running on 4-way SMP 500 MHz Pentium III system with 2 GB of RAM
- 32 machines of a similar configuration were used for load generation

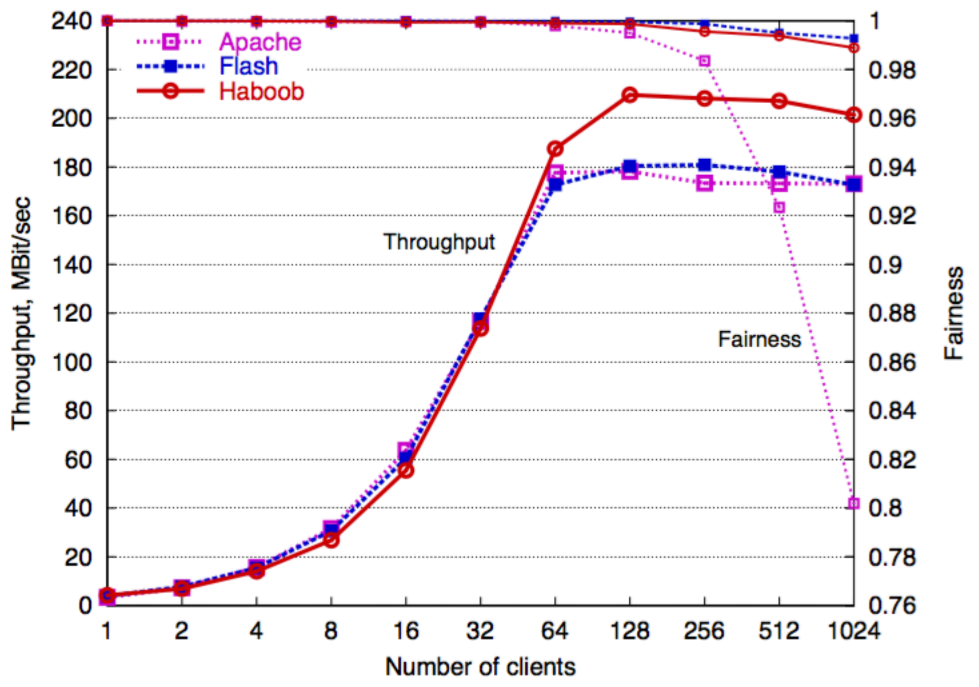
## Other designs

- Apache Web server
  - Thread-based concurrency
  - Fixed-size process pool of 150 processes

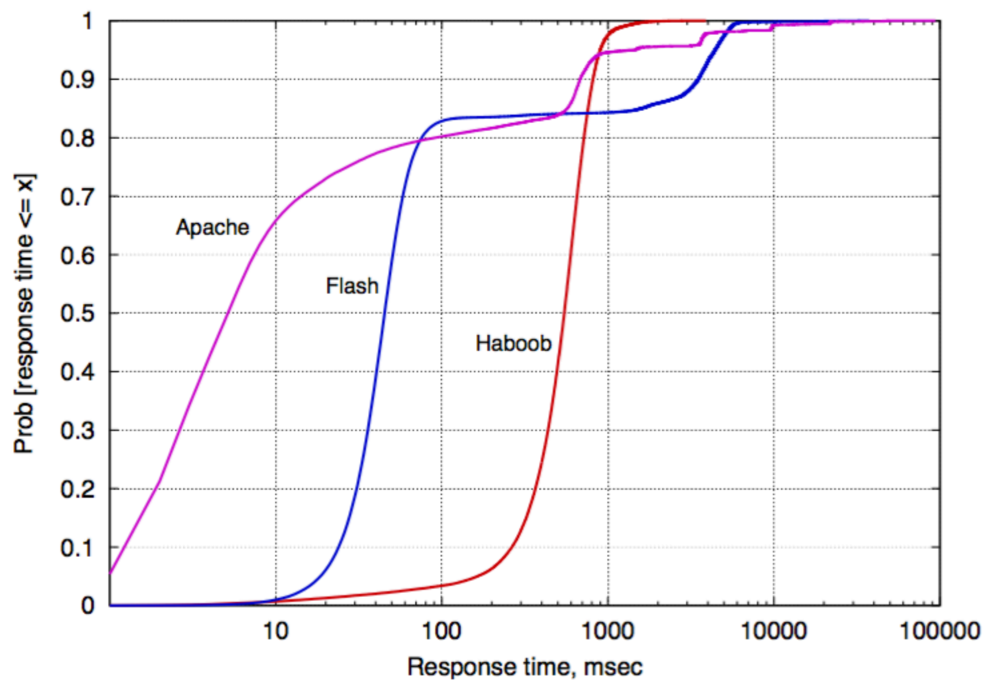
- Flash Web server
  - Event-based concurrency
  - Single process handling most request-processing tasks.
  - Up to 506 simultaneous connections (due to limitations of select system call).

- Haboob
  - Hybrid approach. Event-based and thread-based concurrency
  - Up to 1024 concurrent requests

# Evaluation

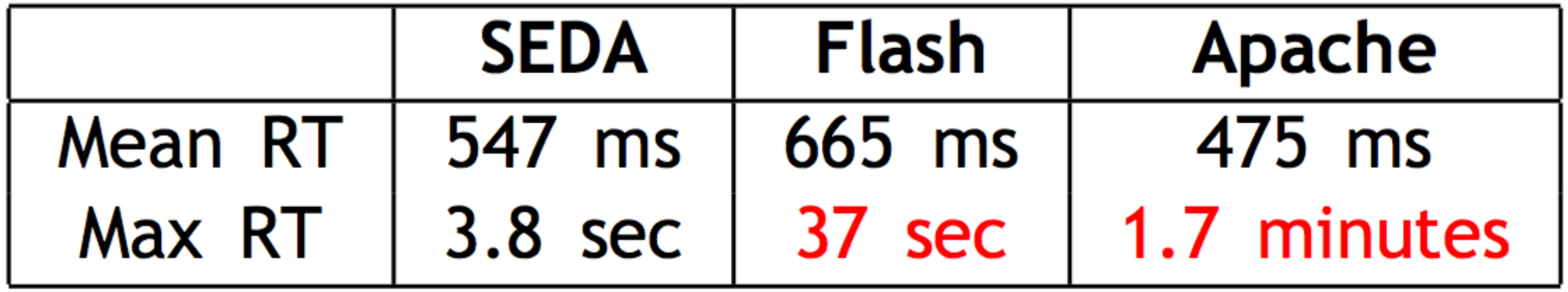


(a) Throughput vs. number of clients



(b) Cumulative distribution of response time for 1024 clients

## Evaluation

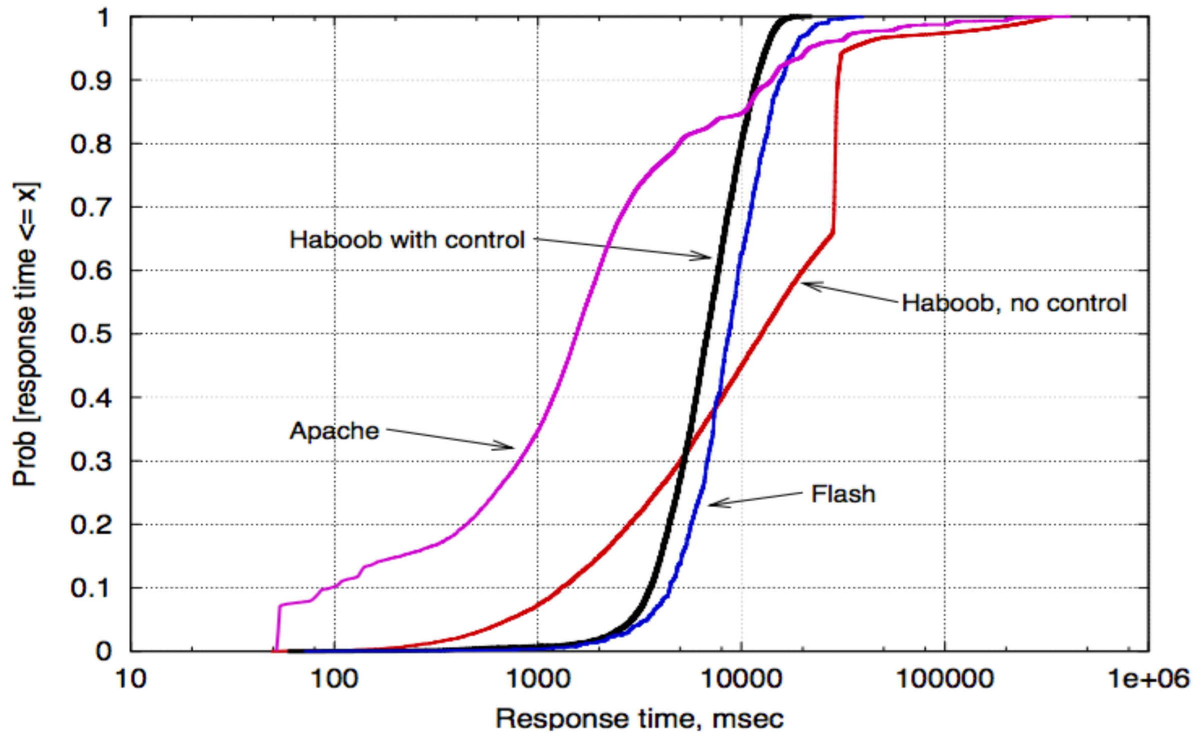


| | SEDA | Flash | Apache |
|---|---|---|---|
| Mean RT | 547 ms | 665 ms | 475 ms |
| Max RT | 3.8 sec | 37 sec | 1.7 minutes |

- SEDA provides some fairness
- The other techniques suffer from long TCP retransmit backoff times. Requests rejected and re-submitted

## Evaluation

- Under overload
  - Requests with high computation and I/O needs from 1024 clients
- Admission control policy by queues
  - Can perform prioritization or filtering during heavy load
  - Adjust size of queue according to the response time
  - Maintain low response time



## Summary

- New designs needed for the ever increasing demands of web services
- SEDA is thus proposed to reach the desired performance
- Combines thread-based and event-based concurrency models
- Splits an application into a network of stages with event queues in between
- Dynamic resource controllers for each stage
- Simplified building high-concurrent services by decoupling load management from core application logic

## Strengths

- High concurrency. Ability to scale to large numbers of concurrent requests
- Ease of engineering. Simplify construction of well-conditioned services
- Modularity. Each stage implemented and managed independently
- Adaption to load variations. Resource management adjusted dynamically.
- Low variance in response time

## Weaknesses

- Increased latency. A request traverse many stages and experiences multiple context switches and additional delays due to queuing.
  - On a lightly loaded server, the worst case context switching overhead can dominate

- What about the average case performance?

  Is the worst response time the most important metric to consider?

- Programming still harder than thread-based concurrency models

Thank you