SEDA

컴퓨터 구조

강우석, 최종우

Table of Contents

Background Concepts

- **User Level Threads vs Kernel Level Threads**
  - User level threads has better performance?!
  - Flexibility
  - Concurrency
  - Virtualized processor

Thread Based Concurrency

- **Special Aspects of Server on Threads.**
  - Application cannot control kernel priority: live lock, starvation

- **Solution :**
  - Exposing more control to applications
  - Augment limited operating system interface
  - Scheduler activation
  - Application-Specific Handler
  - SPIN
  - Exokernel

Bounded Thread Based Concurrency

- **Limits or bond thread with other ideas**
  - Resource container : application to associate scheduling information with activity
  - Scout : Mix thread with how to make ”path” through layers.

Event-Driven Concurrency

- **Better scalability**
- **Run event queues with loop**
- **Robust to load, little degradation beyond saturation**
- **Non-blocking I/O is required, but most of prior work has event-processing threads that can block I/O**
- **Scheduling and ordering is key**
- **JAWS :**
  - Analyze Web server performance bottlenecks
  - Studied trade-off between thread and event-driven concurrent models
  - Emphasizes adaptivity in service design

Structured Event Queue

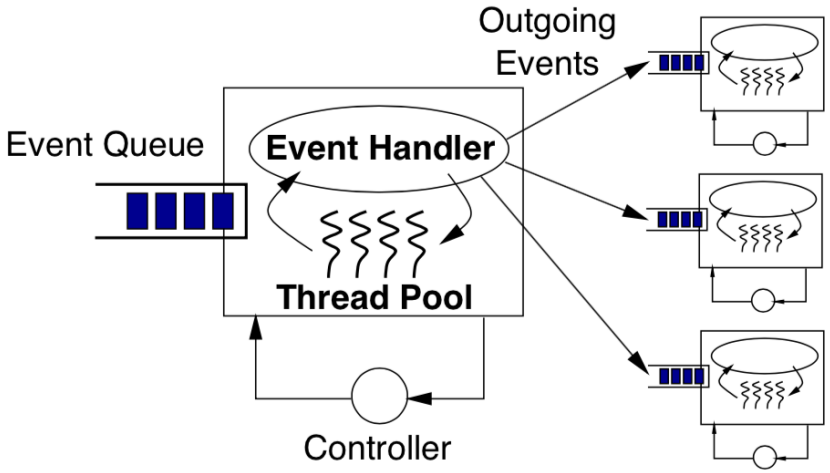
- **To improve modularity and make easier scheduling and ordering, make set of event queues**
- **Queue of two different components are managed separately**
- **SEDA is in this category**
- **Click :**
  - Encapsulates packet processing stages into “Element”
  - Targets specific applications.
  - Single thread serves all event queues
  - Relatively static schedule based on bounded processing time
- **DDS**
  - Emulates asynchronous network and disk I/O interface with fixed size of thread pools
  - Software components are composed with event queue or upcalls
- **Other examples:**
  - Work Crew(1989), TSS/360 Scanner(1968)

Implement

- **Sandstorm**
- **Made in Java**
- **To improve performance, fine tuning has done.**

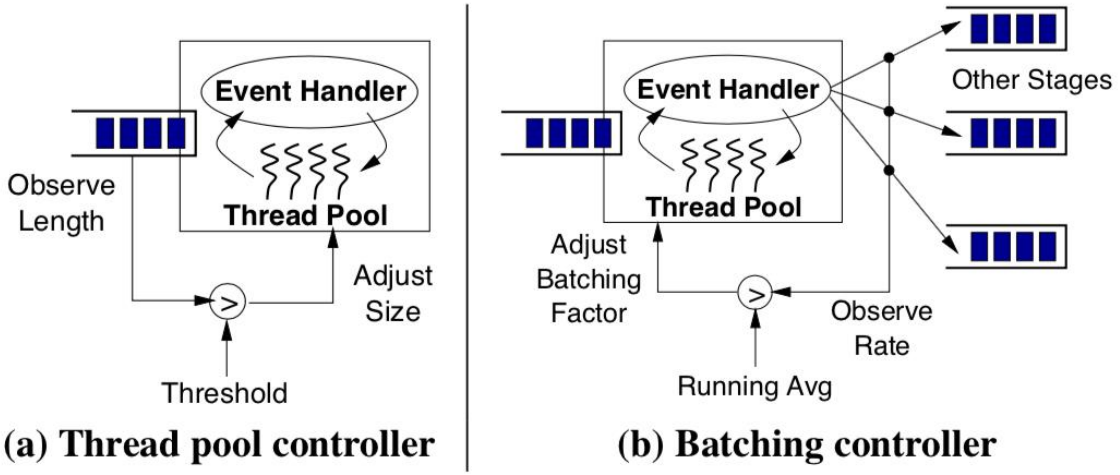
Stages

- **Stage is made up of event queue, thread pool, event handler.**
- **Resource controller allocates and schedules system dynamically.**



Dynamic Resource Controllers

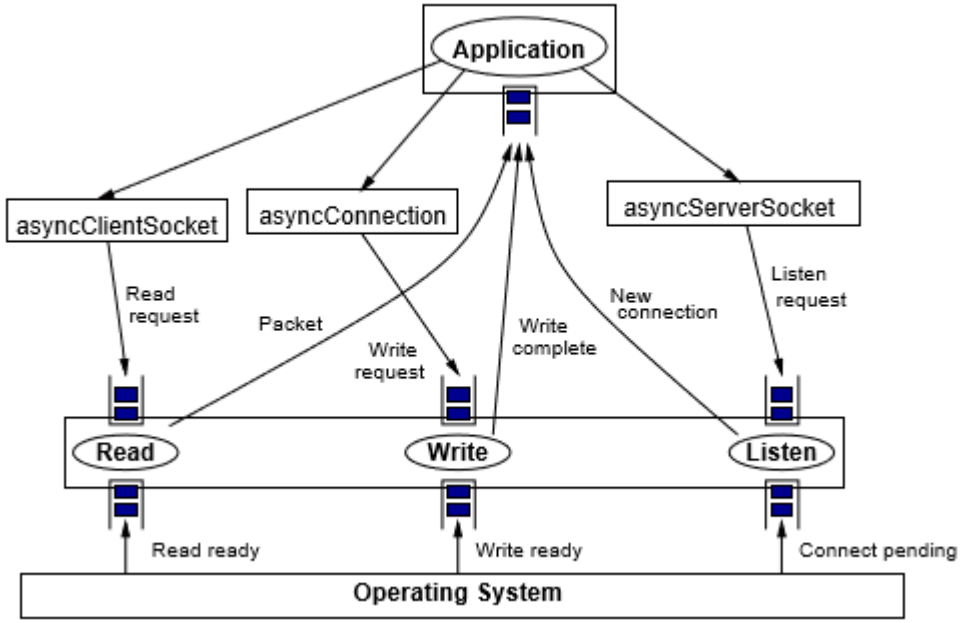
- **Controller that adjusts its resource allocation**
  - **Thread pool controller adjusts the number of**
  - **Batching controller adjusts the number of events processed by each iteration**



Asynchronous socket I/O

Composition

- 3 interfaces : asyncClientSocket, asnycServerSocket, asyncConnection
- 3 stages : Read stage, Listen stage, Write stage
- 2 queues : from user, from OS



Asynchronous socket I/O

Detail

- **Queue**
  - Timeout mechanism을 이용하여 두 queue를 번갈아 서비스
  - Library 형태로 구현되어 동작 시 OS call을 부름
    - UNIX poll과 /dev/poll 지원
  - Randomize order
    - 목적 : Fairness의 확보
    - Reordering 이유 : OS는 보통 socket event를 고정된 순서로 제공

Asynchronous socket I/O

Detail

- **Read stage**
  - I/O readiness event가 데이터를 가진 이용 가능한 소켓이 있음을 알려줄 때 읽음
  - 각 socket request 당 16KB buffer 할당
  - Java의 garbage collection을 deallocation 시 이용
  - Optional rate controller
    - 기준 : moving average of incoming packet rate
    - 방식 : event-processing loop의 delay
    - 효과 : load-shedding

Asynchronous socket I/O

Detail

- **Write stage**
  - 사용자로부터 request를 받고서 OS가 socket이 이용 가능하다 할 때마다 다음 packet을 씀
  - ‘slow’ sockets 방지
    - ‘slow’ sockets 원인
      : 많은 양의 outgoing packet이 특정 connection에 집중될 때 queue의 length와 메모리 사용량이 지나치게 증가하는 문제
    - 해결 방법
      : 임계를 넘어가는 연결을 끊는 방법 이용

Asynchronous file I/O

Composition

- Blocking I/O와 bounded thread pool로 구성
- asyncFile object를 이용
- read, write, seek, stat, close 인터페이스 제공

How it works

- 한 파일당 하나의 thread 이용
- Thread pool controller가 concurrency수요에 따라 동적으로 thread 제어
- 최대 20개까지 thread 수 증가

Haboob

High-performance HTTP Server based on SEDA

- 10 stages
  - 4 Asynchronous socket & disk I/O
  - HttpParse & HttpRecv & HttpSend
  - PageCache & CacheMiss

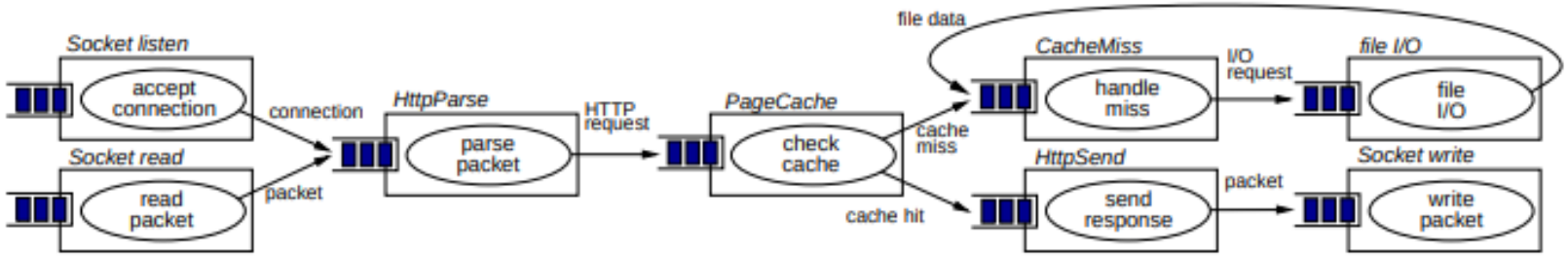
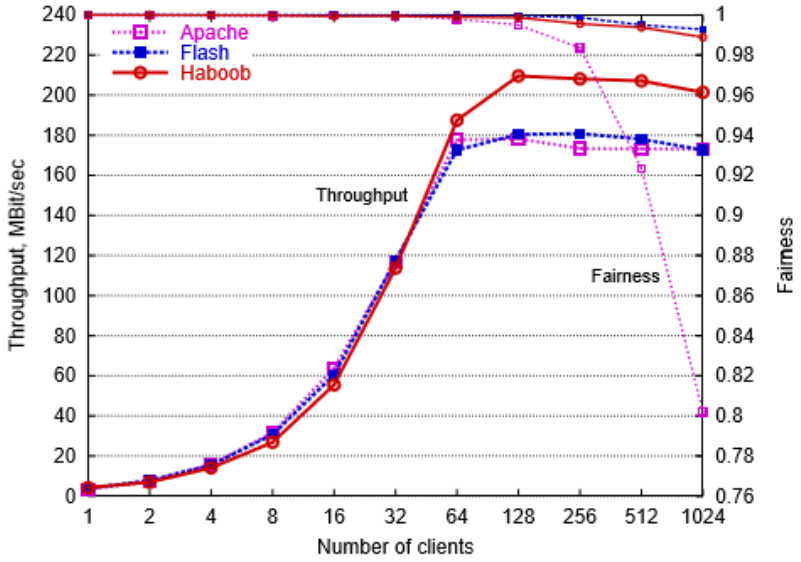
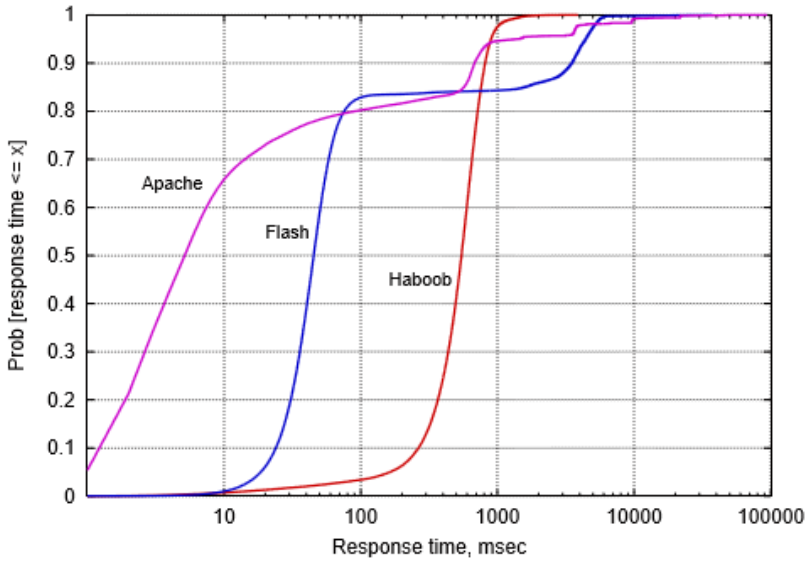


Figure 5: **Staged event-driven (SEDA) HTTP server:** *This is a structural representation of the SEDA-based Web server, described in detail in Section 5.1. The application is composed as a set of* stages *separated by* queues. *Edges represent the flow of events between stages. Each stage can be independently 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.*

Haboob



(a) Throughput vs. number of clients



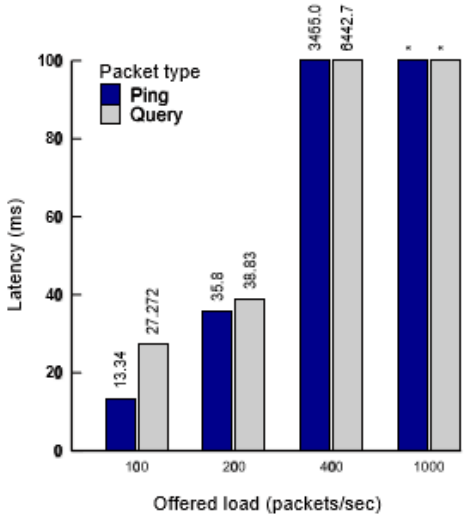
(b) Cumulative distribution of response time for 1024 clients

	256 clients				*1024 clients*			
Server	*Throughput*	*RT mean*	*RT max*	*Fairness*	*Throughput*	*RT mean*	*RT max*	*Fairness*
Apache	173.36 Mbps	143.91 ms	27953 ms	0.98	173.09 Mbps	475.47 ms	93691 ms	0.80
Flash	180.83 Mbps	141.39 ms	10803 ms	0.99	172.65 Mbps	665.32 ms	37388 ms	0.99
Haboob	208.09 Mbps	112.44 ms	1220 ms	0.99	201.42 Mbps	547.23 ms	3886 ms	0.98

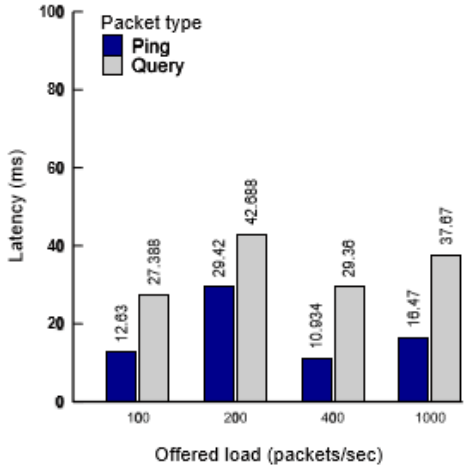
Gnutella packet router

Gnutella

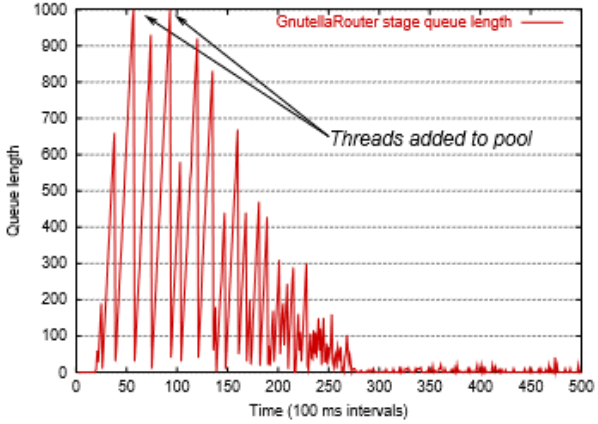
- Peer-to-peer network service
- Asynchronous socket I/O + 3stages(GnutellaServer, GnutellaRouter, GnutellaCatcher)



(a) Using single thread



(b) Using thread pool controller



(c) Queue length profile

Conclusion

Main Contribution

- Stage를 통해 Modularity를 확보하여 debugging이 쉽고 scalable하며 비교적 application 작성이 쉬운 인터넷 서비스 디자인을 제시
- Controller를 통해 request간 fairness를 확보하여 predictable하게 response time이 감소하는 서비스 제공

Expectation

- Multi-machine 환경에서도 추가적인 resource virtualization 없이 서비스를 구현할 수 있을 것으로 기대
- 다양한 scheduling policy가 개발 될 것으로 기대

Pro & Con

Pro

- Focused, application-specific admission control
- Exposure of the request stream
- Modularity and performance isolation

Con

- Application 개발자에게 여전히 구현 부담
- Queueing과 context switch의 overload가 크다
- Java의 garbage collection 기능은 high-performance 시스템에 적합하지 않다

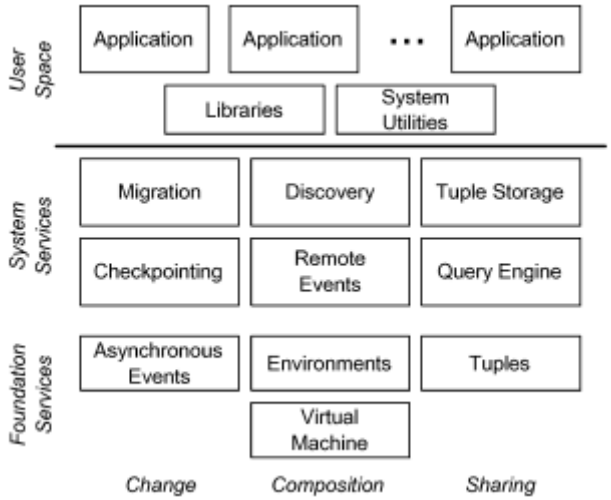
Cons.

System Support for Pervasive Applications

ROBERT GRIMM et al.

- SEDA의 2가지 문제점 언급
  - Application을 Stage로 구성해야 하는 어려움
  - Stage간 event queueing 방식을 Application 개발자가 직접 작성해야 한다는 어려움

- one.world (integrated framework for building adaptable applications) 제안

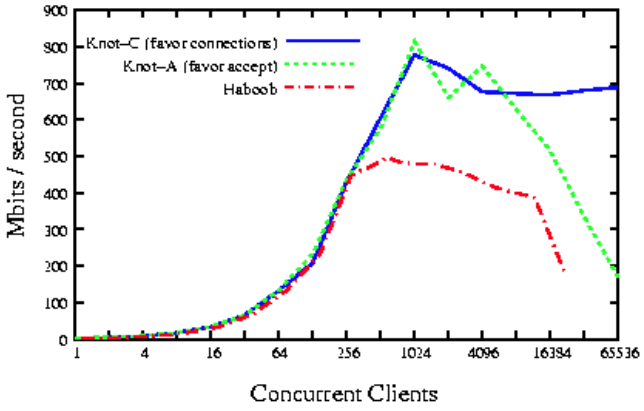


Cons.

Why Events Are A Bad Idea (for high-concurrency servers)

Rob von Behren, Jeremy Condit and Eric Brewer

- Thread 시스템의 정당성 논증
- 실제 Haboob과 Knot의 비교를 통해 SEDA의 문제점 지적
  - Haboob 성능 저하 추정 원인
    - Stage마다 thread pool 운용으로 결국 전체적으로 굉장히 많은 context switch 존재
    - 다수의 crossing과 queuing operation 존재
    - Java의 garbage collection이 성능 저하를 일으킴



Subsequent Thesis

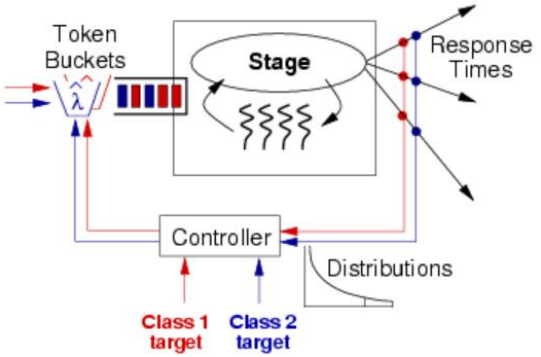
Adaptive Overload Control for Busy Internet Servers

Matt Welsh and David Culler

- 동적으로 Overload를 조절하는 방법 설명
  - Metric : 90th-percentile response time
  - 동작 :
    - Monitor : Response time 측정
    - Controller : admission control parameter 조절

- 논문에서는 Token Bucket의 token이 생성되는 rate를 조절하는 방식으로 동작하나 다양한 조절 정책이 적용될 수 있음을 언급

- Priority에 기반한 class-based service level agreements와 관련된 여러 정책을 적용할 수 있음을 언급



Capriccio: Scalable Threads for Internet Services

Rob von Behren, Jeremy Condit, Feng Zhou, George C. Necula and Eric Brewer

- High Concurrency를 지원하는 Thread 기반의 아키텍처 제시
  - User-level thread 이용

Appendix : Think more

Agile Dynamic Provisioning of Multi-Tier Internet Applications

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal and Timothy Wood

- SEDA가 provisioning issue를 고려하지 않았음을 언급
  - Dynamic Provisioning을 Multi-tier 프로그램에 적용하면 heavy workload에서 좀 더 안정적인 서비스 환경 제공을 생각해볼 수 있음

A method for transparent admission control and request scheduling in e-commerce web sites

Sameh Elnikety, Erich Nahum, John Tracey and Willy Zwaenepoel

- 간접적으로 overload를 측정하는 SEDA와 달리 스크립트당 리소스의 사용을 측정
  - 좀 더 정확하고 세분화된 조절을 할 수 있음

Reference

Matt Welsh, David Culler, and Eric Brewer, “SEDA: An Architecture for Well-Conditioned, Scalable Internet Services,” SOSP, 2001.

Welsh, Matt and David E. Culler. “Adaptive Overload Control for Busy Internet Servers.” *USENIX Symposium on Internet Technologies and Systems* (2003).

Behren, J. Robert von et al. “Why Events Are a Bad Idea (for High-Concurrency Servers).” *HotOS* (2003).

Grimm, Robert et al. “System support for pervasive applications.” *ACM Trans. Comput. Syst.* 22 (2003): 421-486.

Urgaonkar, Bhuvan et al. “Agile dynamic provisioning of multi-tier Internet applications.” *TAAS* 3 (2008): 1:1-1:39.

Elnikety, Sameh et al. “A method for transparent admission control and request scheduling in e-commerce web sites.” *WWW* (2004).

Behren, J. Robert von et al. “Capriccio: scalable threads for internet services.” *SOSP*(2003).