Perspectives on Threaded and Asynchronous Programming

Oliver Kennedy

Advanced Systems

Lecture 2

Threaded vs Asynchronous

- *CPUs are faster than most hardware*
- *Programs need to wait for hardware*
- *How does a program continue processing while waiting on hardware?*

Poll and Process

- *Continually poll the hardware for readiness*
- *Extremely inefficient*

Threads

- *Multiple threads processing data at once*
- *While one thread waits, the rest continue*
- *Benefits*
  - *Intuitive, Multiprocessor Support, Modular code can't break the system*
- *Drawbacks*
  - *Inefficient, Synchronization issues*

Asynchronous Programming

- *One thread processes events*
- *Hardware readiness treated as an event*
- *Benefits*
  - *Efficient, Linear Execution, No Starvation*
- *Drawbacks*
  - *Unintuitive, Problems with Modular Code, Limited Concurrency, Tasks need to be short*

Threads in Interactive Systems

- *Programmers use threads a lot*
- *A lot of modern problems have to do with threads*
- *What can we learn from this?*
  - *What do programmers use threads for?*
  - *What mistakes are made with threads?*
  - *How can threads be made more efficient?*

Cedar and GVX

- *A case study on two OSes*
- *Cedar and GVX use the Mesa language's thread system*
  - *Mesa supports standard primitives*
  - *... and a strict priority scheduler*

And they found...

- *A bird's eye view: The Profiler*
- *Three classes of threads*
  - *Eternal Threads*
  - *Worker Threads*
  - *Transient Threads*
- *Cedar vs GVX*
  - *Free and loose vs Small and Efficient*

Thread Paradigms

- *Defer Work*
- *Pumps/Slack Processes*
- *Sleepers/One Shots*
- *Deadlock Avoiders*
- *Task Rejuvenation*
- *Serializers*
- *Concurrency Exploiters*
- *Encapsulated Forks*

Thread Problems-What works

- *Sleepers, One shots, Pumps, Work-Deferrers all implemented properly*
  - *Yet these require little inter-thread interaction*
- *Concurrency Exploiters were new at the time*
  - *Work has been done since*

Problems-Time Constraints

- *High priority slack-processes can be hard to write for use with low priority threads*
  - *Yield and a strict priority scheduler don't play nice*
- *Solution: Add a YieldButNotToMe primitive*

Problems-Priorities

- *Synchronization primitives cause priority inversions in strict-priority schedulers*
- *Solution 1: High priority threads donate cycles to threads holding locks they need*
- *Solution 2: A high priority thread that periodically grants a time slice to a thread chosen at random*

Problems-Misunderstandings

- *Mesa implements locks in an unusual way*
  - *Programmers write code that might be correct in some circumstances*
- *Bugs introduced this way are hard to track (the code looks right)*

Problems-Treating the Symptom

- *A common problem*
- *Ex: Fixing a wait without a corresponding notify by adding a timeout to the wait*
- *Introduces delays and possibly bugs*

Problems-Changing Hardware

- *Magic Numbers*
  - *Timeouts and pause lengths based on one processor become invalid when a faster processor comes out*
- *Memory Ordering*
  - *Much code assumes strict memory ordering*

Problems-Library Implementation

- *Notify, Yield, and Scheduling*
- *Strict priority scheduling sucks*
- *Time quantum*
  - *(not a problem, but a consideration)*

Future Work

- *Analyze more systems!*
- *Come up with new scheduling techniques*
- *Keep analyzing known code*

SEDA

- *The internet is big... really big*
- *Loads are getting bigger*
- *Dynamic content becoming prevalent*
- *Services need to adapt to these loads*

But how can we adapt?

- *Can't we use threads?*
- *What if we only used so many threads?*
- *Weren't you talking about some asynchronous nonsense earlier?*
- *How about a mix?*

SEDA

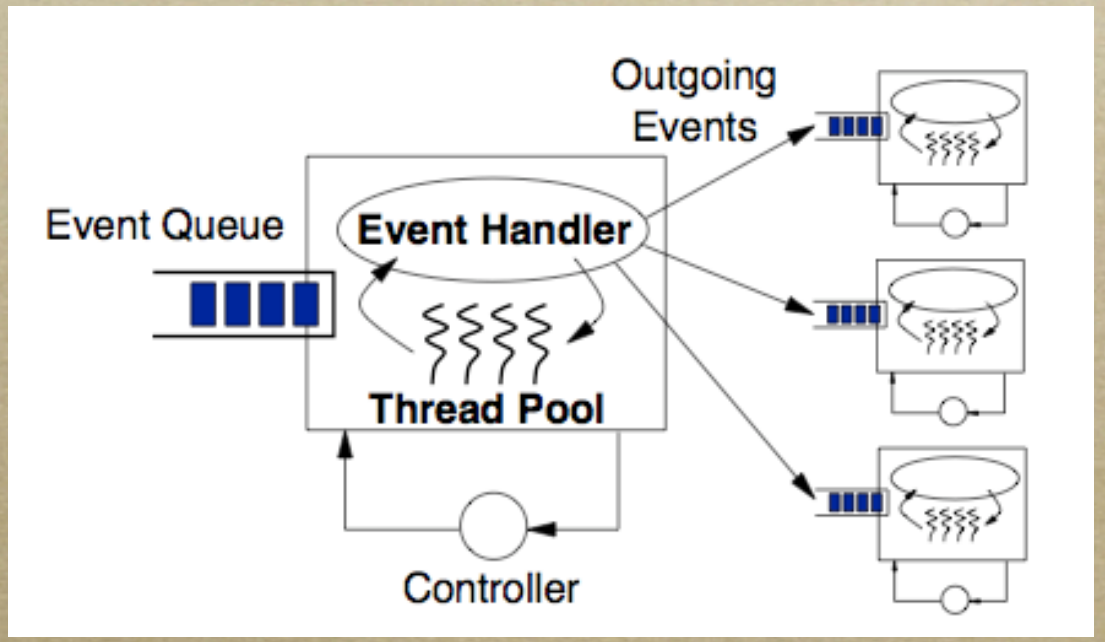
- *A means of building scalable web services*
  - *Has to support concurrency*
  - *Has to be easy to program*
  - *Has to let the application manage load*
  - *Has to tune itself*

Stages

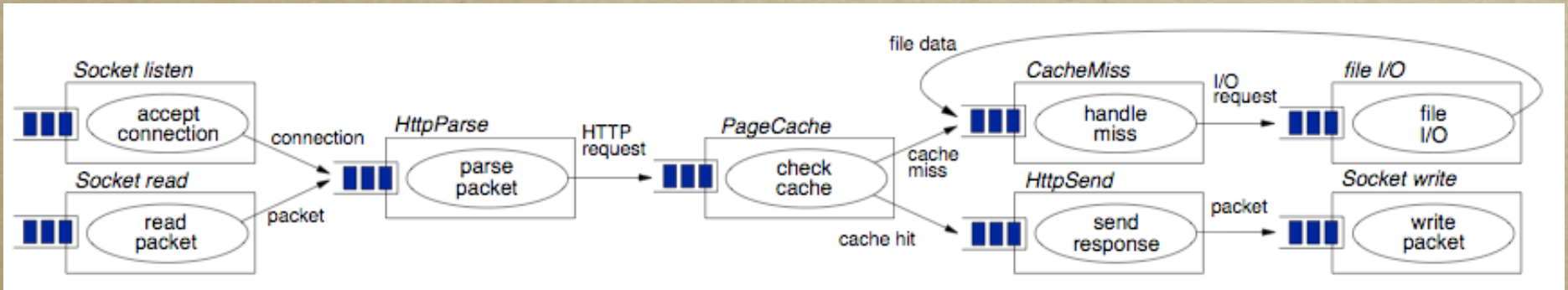
- *Any task can be broken down*
- *SEDA breaks tasks down into stages*
- *A stage has an input queue*
- *A full web-service is multiple stages networked together*

Stages

- *Event Queue*
- *Thread Pool*
  - *Automatic tuning*
- *Controller*
  - *Feedback*
- *Event Handler*



Stages


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

Why stages?

- *Allows for isolation*
- *Fine grained tuning*
- *Easier debugging*

Self-Tuning

- *Each stage has an associated controller*
- *Thread Pool Size*
  - *More threads = More concurrency (up to a point)*
- *Request Batching*
  - *Cache Locality, Task Aggregation*

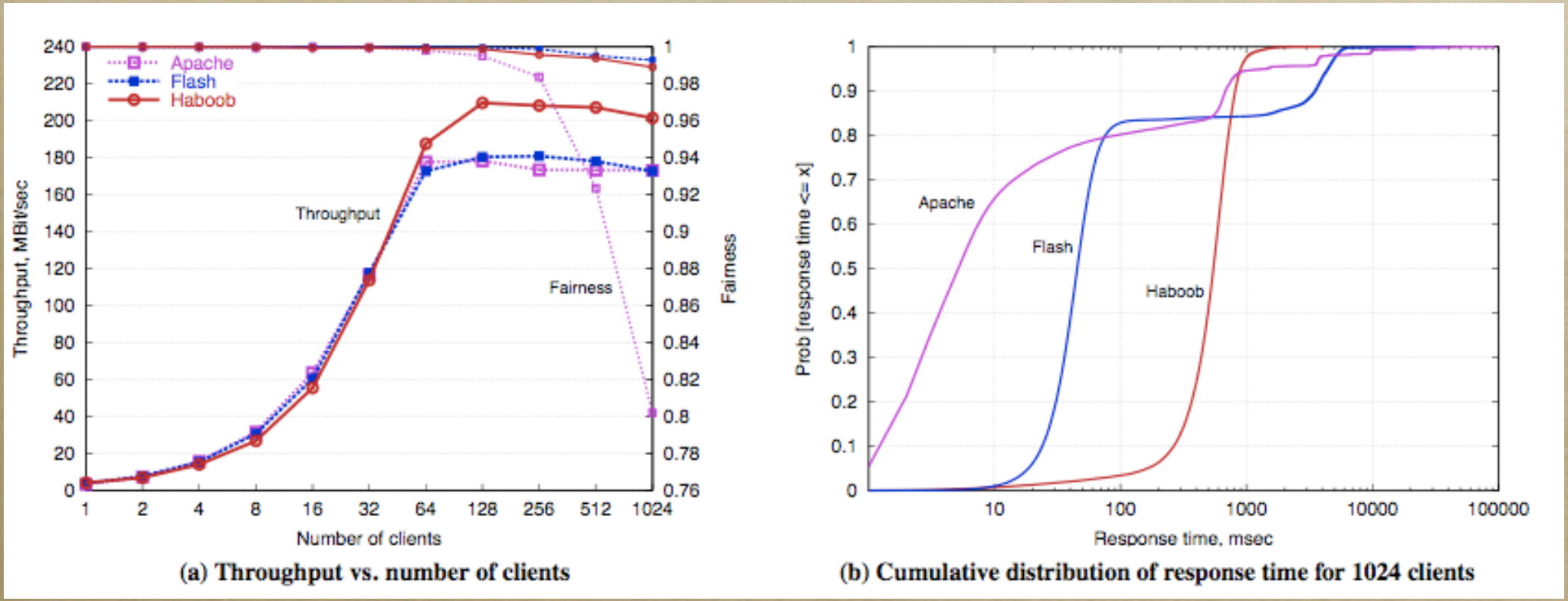
Sandstorm

- *Java based implementation of SEDA*
  - *Simple memory management*
- *Provides APIs*
  - *Queue and stage management*
  - *Profiling/debugging*

Haboob/GnutellaServer

- *An implementation of common web services on top of Sandstorm*
- *Both performed admirably*
- *Haboob (despite being written in Java) had better performance characteristics than Apache under high loads.*

Haboob vs Apache vs Flash



(a) Throughput vs. number of clients

(b) Cumulative distribution of response time for 1024 clients

Threads vs Hybrids

- *The age old conflict*
  - *Monolithic vs Microprogramming*
- *Threads expose more*
  - *It's 12 years later, the tech is here*
- *SEDA is more elegant*
  - *But it's in Java...*

Any Questions?