Virtualization Considered Harmful:
OS Design Directions for Well-Conditioned Services

Matt Welsh and David Culler

UC Berkeley Computer Science Division
mdw@cs.berkeley.edu

`http://www.cs.berkeley.edu/~mdw/proj/seda`

Motivation: Internet Services

Massive concurrency demands

- Yahoo: 1.1 *billion* pageviews/day
- Gartner Group: 127 million adult Internet users in US

Must be extremely robust to load

- Peak load many times that of average
  - ▷ *Overprovisioning is generally infeasible*
- Recent Presidential Election: 130% - 500% increase in news site traffic
- Load spikes occur exactly when the service is most valuable!

Engineering robust, scalable services is difficult

- Much work on fast static Web servers (caching, replication)
- But services are increasingly dynamic:
  - ▷ *e-Commerce, stock trading, driving directions, etc.*
- The future: *open services*

The Problem: Resource Virtualization

Existing OS designs strive to virtualize resources

- Thread/process-based concurrency is expensive
- Applications rarely given opportunity to affect policy
- Extensible OSs (SPIN, Exokernel, etc.) a step in the right direction
  - ▷ *But still based on multiprogramming mindset*

## We propose SEDA, a new design for robust, scalable services

- Construct service as set of *stages* connected by explicit *event queues*
- Refinement of event-driven model leading to high concurrency
- Exposes request stream to application
- Facilitates dynamic resource control

Our claim: Server OS's should abandon transparent virtualization

- Internet services do not need a general-purpose OS
- Rather, require massive concurrency and resource control
- Virtualization makes this difficult (if not impossible) to achieve

# Evil #1: Concurrency Limitations

- Processes and threads designed for *timesharing*
    - ▷ *Entail high overheads and memory footprint*

- These mechanisms don't scale to many thousands of tasks
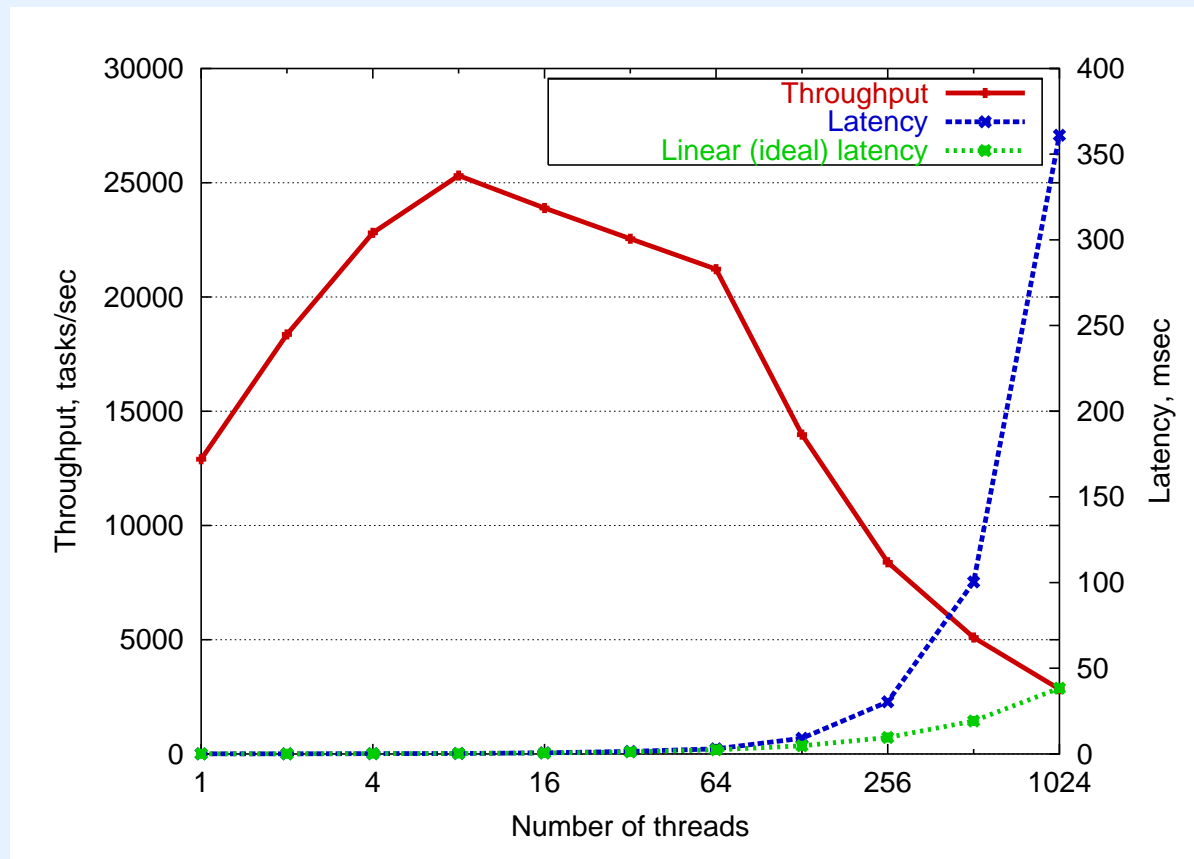
(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

Evil #2: I/O Scalability Limitations

- Blocking I/O requires many threads for concurrency
- Data copies (an artifact of virtualization) are a bottleneck
- Degrade in performance as number of I/O flows increase
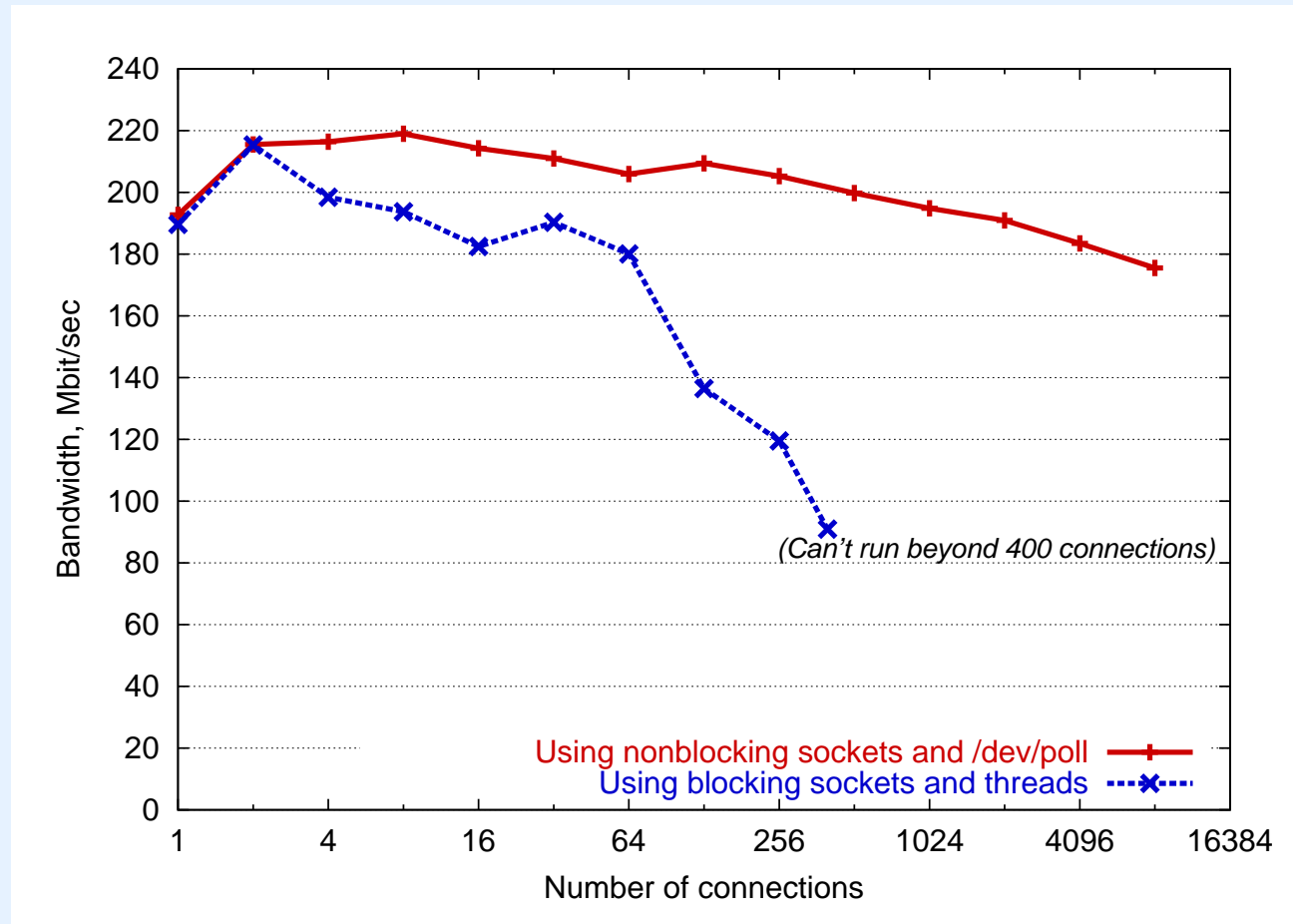
Evils #3 & 4: Transparent resource management and Coarse-grained Scheduling

Virtualization implies request satisfaction regardless of cost

- O/S hides performance aspects of interfaces
- e.g., Request to allocate memory that requires paging
- Leads to thrashing when resources are overcommitted

Thread-based concurrency yields little control over scheduling

- Only knobs are prioritization or runnable status of threads

Desire control over flow of requests through a resource

- e.g., Reorder request stream through Web page cache
- Prioritize cache hits over misses, reject if overloaded
- Very difficult to do this via thread scheduling

Traditional Event-Driven Programming
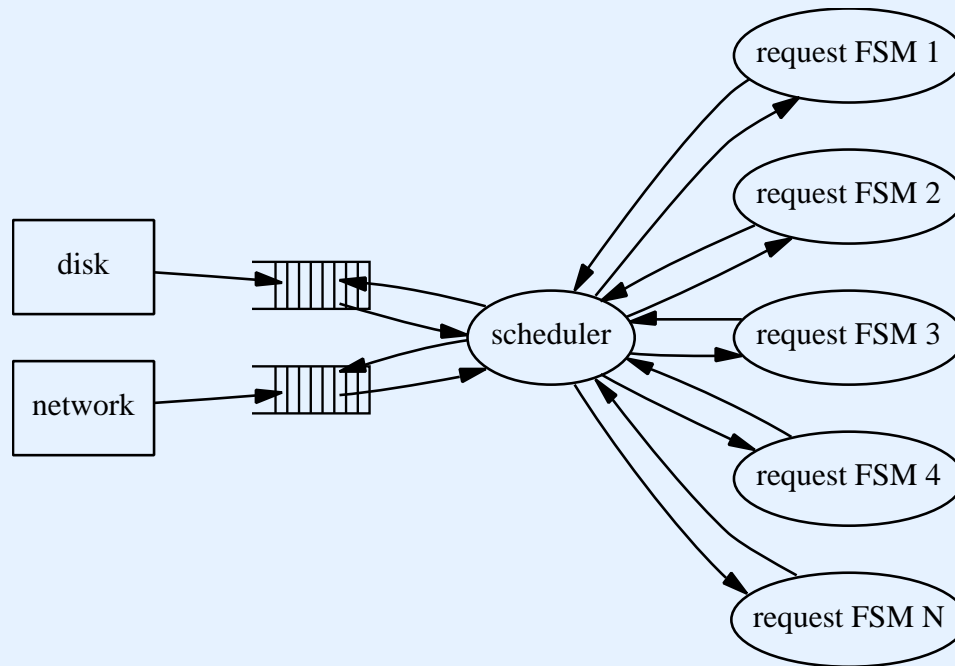
- Many designers eschew threads and roll their own event-processing system
- Each concurrent flow implemented as a finite state machine
- Difficult to construct, modularize, maintain, and debug
    ▷ *Requires careful scheduling of FSMs and events*
    ▷ *Resulting structure is often application-specific and brittle*

Staged Event-Driven Architecture (SEDA)
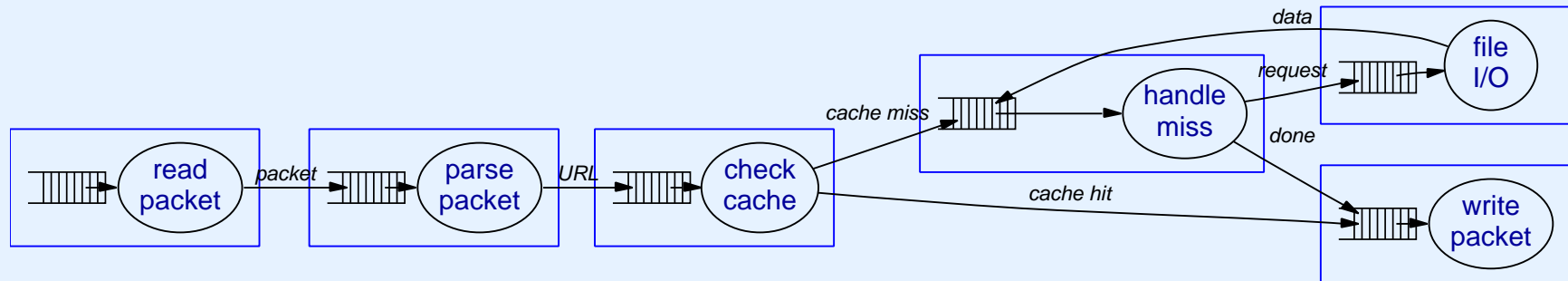
Decompose service into *stages* separated by *queues*

- Stages embody a set of states from the event-driven FSM
- Queues introduce control boundary, absorb excess load

Stages contain internal *thread pool* to drive execution

- Stages may run in sequence or in parallel
- Thread allocation and scheduling managed by *stage controller*
- Stages may block internally: Thread pool grows with demand

SEDA Benefits

Support for high concurrency

- Use of small number of threads and nonblocking I/O scales well
- Number of threads chosen at a per-stage level by controller
 - ▷ *Avoids wasting threads on stages with little concurrency*

Application-specific load conditioning

- Explicit event queues expose request streams
- Backpressure implemented by blocking on full queue
- Load shedding implemented by dropping events
  - ▷ *May also take alternate action, e.g., degraded service*

Improved code modularity and debugging

- Stages can be developed independently and composed
- Tracing event propagation and queue lengths facilitates debugging

SEDA at Work: SPEC Web99 Performance
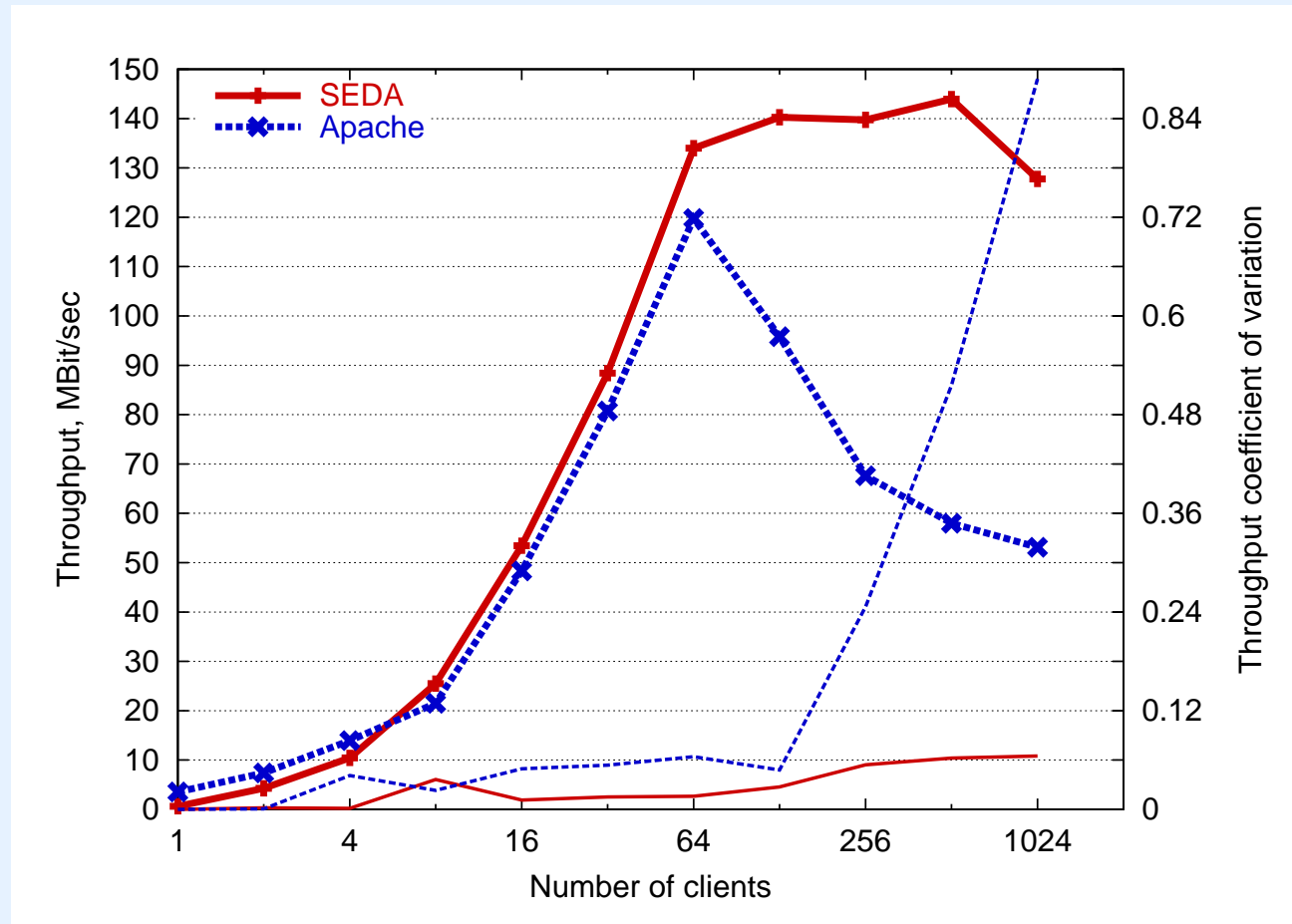
- SEDA performance stable for large number of connections
  - ▷ *Some degradation due to Linux socket inefficiencies*
- Apache degrades noticeably, exhibits very poor fairness

Designing an OS for Internet Services

SEDA model suggests several avenues for OS research

- We argue SEDA is the right model for building well-conditioned services
- Still, OS designers should reevaluate the goal of virtualization
- More radical than safe extensibility: *Need not be ''safe!''*

Reconsidering the role of safety

- SEDA-based OS need not support multiple apps safely sharing resources
- Internet services are highly specialized and do not share machine
- Can still support *protection* without masking resource *availability*

Concurrency and scheduling support

- Rather than do away with threads, reduce dependence for concurrency
  - ▷ *OS interfaces should be uniformly nonblocking*
- Application should be able to specify scheduling policy
  - ▷ *e.g., Give priority to stages that free resources*
  - ▷ *Delay scheduling of stage until it has accumulated work*
- System can trust this policy, since safety is not an issue

More OS Design Directions

Scalable I/O support

- Goal: Support large number of I/O streams through one app
  - ▷ *Rather than safely multiplex I/O across many apps*
- Claim: No virtualization makes it easier to support scalability
- Expose I/O readiness and completion events directly
  - ▷ */dev/poll, kernel event queues, etc. are the right model*
  - ▷ *select() etc. are an afterthought and do not scale*
- Facilitates zero-copy I/O (which is difficult to virtualize)

Application-controlled resource management

- Application hinting and control is vital for load management
- OS should expose low-level interfaces rather than generic abstractions
  - ▷ *Similar philosophy to Exokernel and SPIN*
- e.g., expose low-level disk interface rather than filesystem/buffer cache
- Virtual memory management should be explicit
  - ▷ *''Scheduler activations'' for generic resources*

New Way of Thinking about Software

Support for massive concurrency requires new design techniques

- SEDA introduces service design as a *network of stages*
- Design for robustness and adaptivity, rather than best case
- Expose request streams to applications for load conditioning

Resource throttling to keep stages within operating regime

- Adapt behavior at runtime to deal with changing load
- Controllers shield service developers from much of this complexity
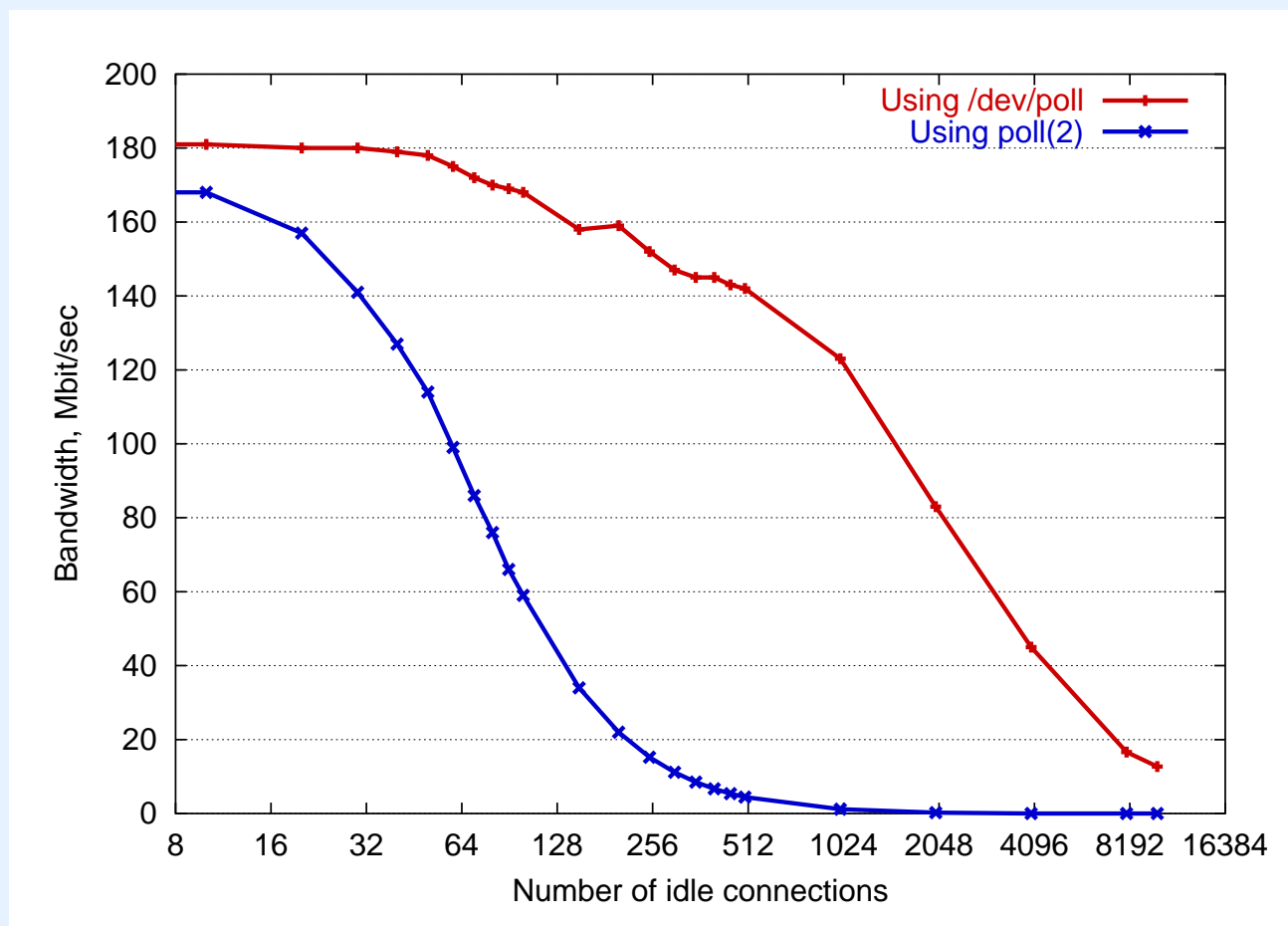
Implications for OS and language design

- SEDA model opens up new questions in OS design space
- Resource virtualization is a ''feature'' that services can do without
- Bring body of work on control systems to bear on service design

For more information

`http://www.cs.berkeley.edu/~mdw/proj/seda`
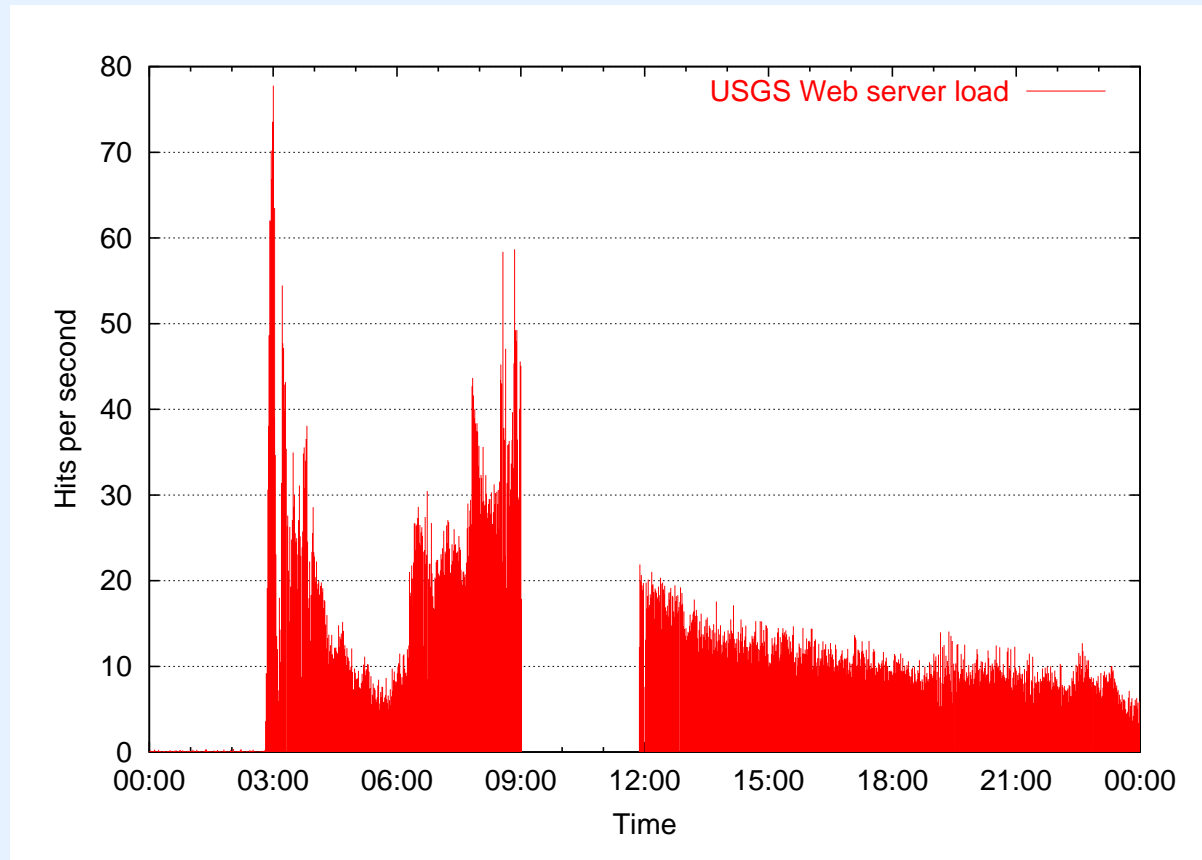
Backup Slides Follow

Effect of Idle Connections

(4-way 500 MHz PIII, Gigabit Ethernet, Linux, IBM JDK 1.1.8)

- One active connection, 1-10000 idle connections
- Compare poll(2) to /dev/poll event dispatch

The Slashdot Effect

Web log from USGS Pasadena Field Office

- M7.1 earthquake at 3 am on Oct 16, 1999
- Load increased 3 orders of magnitude in 10 minutes
- Disk log filled up at 9am