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

Welsh01: Matt Welsh, David Culler, Eric Brewer. ACM Symp. on Operating Systems Principles (SOSP-18), October 2001.

Managing Concurrency

- Internet services suffer spikes of lots of work
- Thread switch expensive for memory
  - Fewer threads can be more efficient
  - Need threads for real parallelism (cores)
- FCFS scheduling not Quality of Service
  - Head-of-line blocking (concurrent threads allow bypass of slow operations by fast)
  - Want application control of ordering
    - Load shedding (reject) or change implementation

Classic threading

- Fixed pool of threads pull from RPC requests
  - No control of work to be done, which block etc
  - Pool size limits work in progress, thrashes memory

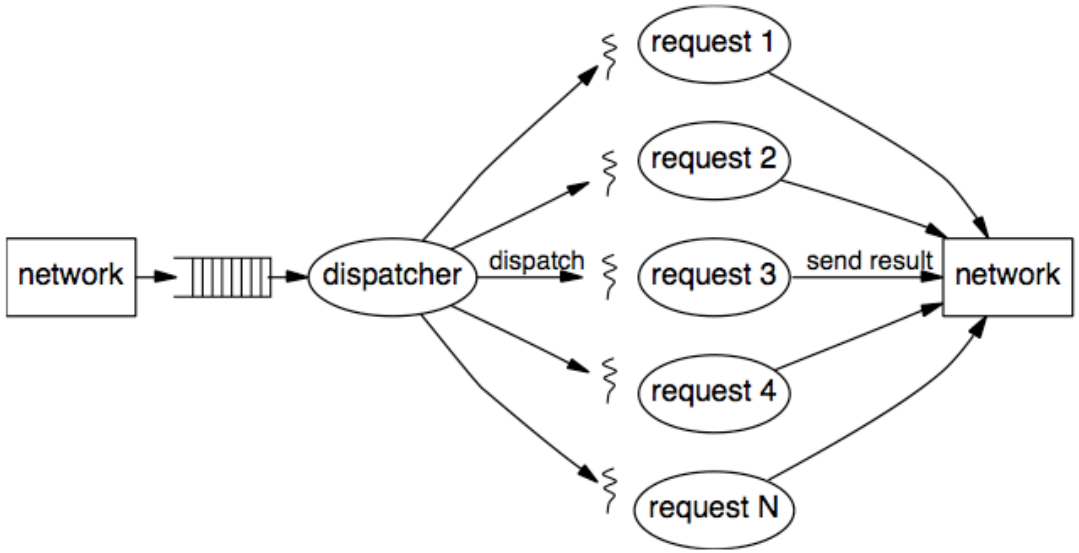


Figure 1: **Threaded server design:** *Each incoming request is dispatched to a separate thread, which processes the request and returns a result to the client. Edges represent control flow between components. Note that other I/O operations, such as disk access, are not shown here, but would be incorporated into each threads' request processing.*

Classic vs event-driven

- Event driven uses minimal threads
  - Receives requests into internal queues and schedules invoking work as subroutines based on knowledge

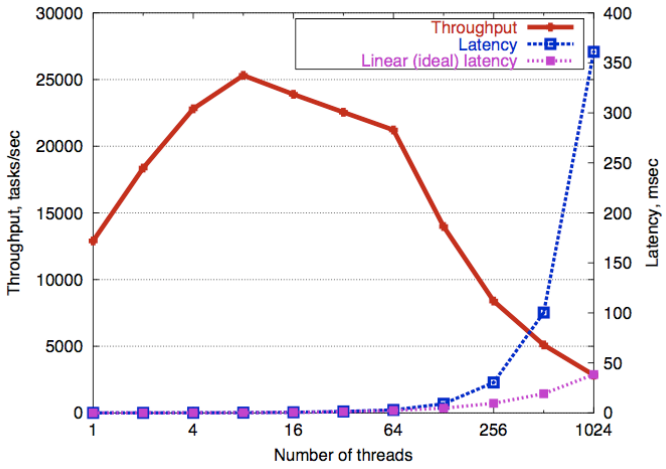


Figure 2: **Threaded server throughput degradation:** *This benchmark measures a simple threaded server which creates a single thread for each task in the pipeline. After receiving a task, each thread performs an 8 KB read from a disk file; all threads read from the same file, so the data is always in the buffer cache. Threads are pre-allocated in the server to eliminate thread startup overhead from the measurements, and tasks are generated internally to negate network effects. The server is implemented in C and is running on a 4-way 500 MHz Pentium III with 2 GB of memory under Linux 2.2.14. As the number of concurrent tasks increases, throughput increases until the number of threads grows large, after which throughput degrades substantially. Response time becomes unbounded as task queue lengths increase; for comparison, we have shown the ideal linear response time curve (note the log scale on the x axis).*

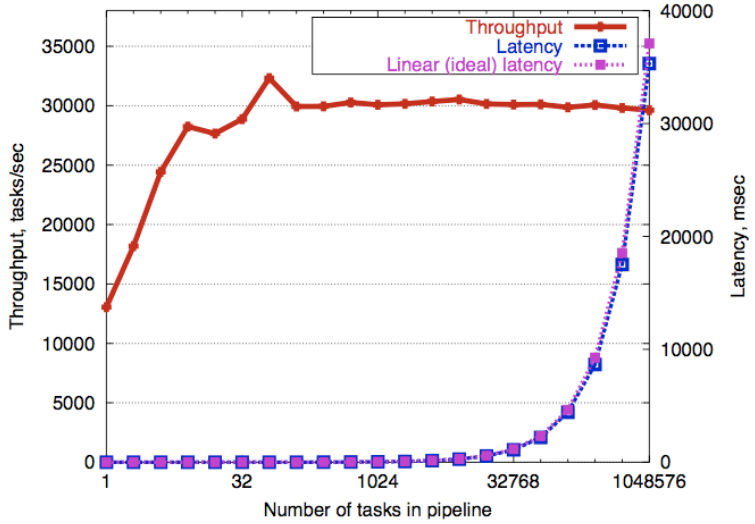


Figure 4: **Event-driven server throughput:** *This benchmark measures an event-driven version of the server from Figure 2. In this case, the server uses a single thread to process tasks, where each task reads 8 KB from a single disk file. Although the filesystem interface provided by the operating system used here (Linux 2.2.14) is blocking, because the disk data is always in the cache, this benchmark estimates the best possible performance from a nonblocking disk I/O layer. As the figure shows, throughput remains constant as the load is increased to a very large number of tasks (note the change in the horizontal axis scale from Figure 2), and response time is linear (note the log scale on the x axis).*

Haboob HTTP server

- Split function into modules with queues between
  - Parse vs page cache – parse sometimes responds w/o calling page cache which can block
  - Same for cache miss and file io
  - Sometimes 3rd party modules have different threading (Berkeley DB for example)
  - Really a messaging system

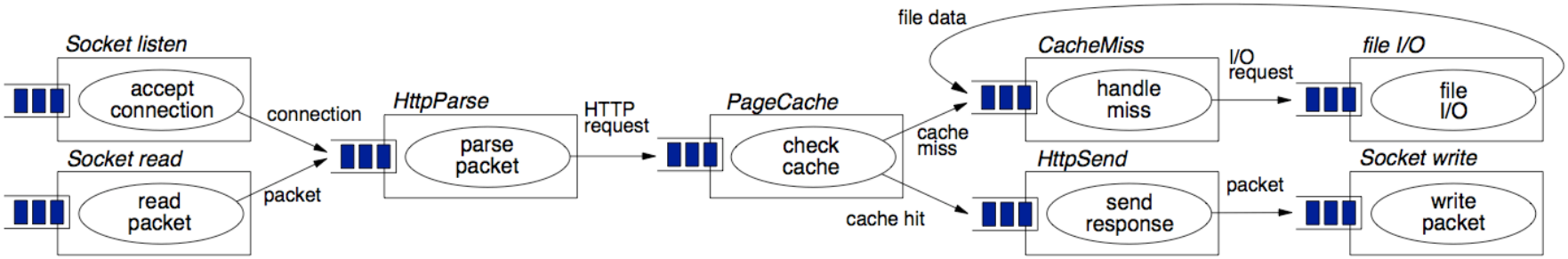
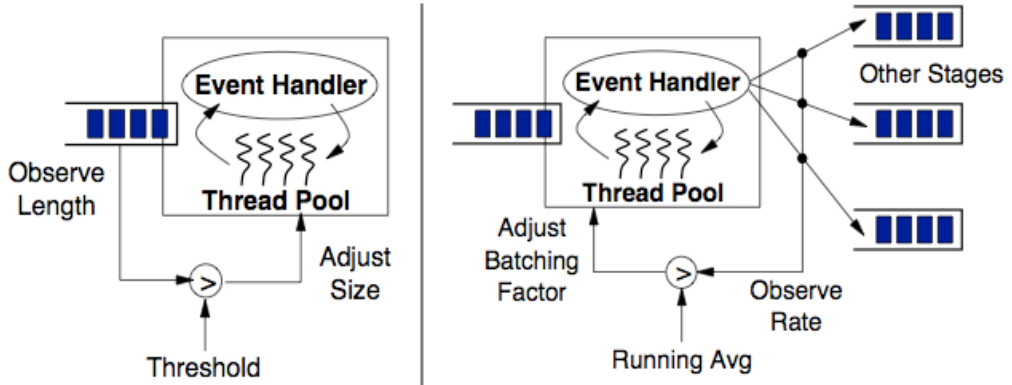


Figure 5: **Staged event-driven (SEDA) HTTP server:** *This is a structural representation of the SEDA-based Web server, described in detail in Section 5.1. The application is composed as a set of* stages *separated by* queues. *Edges represent the flow of events between stages. Each stage can be independently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

Controller for introspection

- Controllers resources (threads, pages if possible)
  - Eg. Based on incoming queue, add concurrencv
  - Based on throughput,
    deepen batch



(a) Thread pool controller **(b) Batching controller**

Figure 7: **SEDA resource controllers:** *Each stage has an associated* controller *that adjusts its resource allocation and behavior to keep the application within its operating regime. The* thread pool controller *adjusts the number of threads executing within the stage, and the* batching controller *adjusts the number of events processed by each iteration of the event handler.*

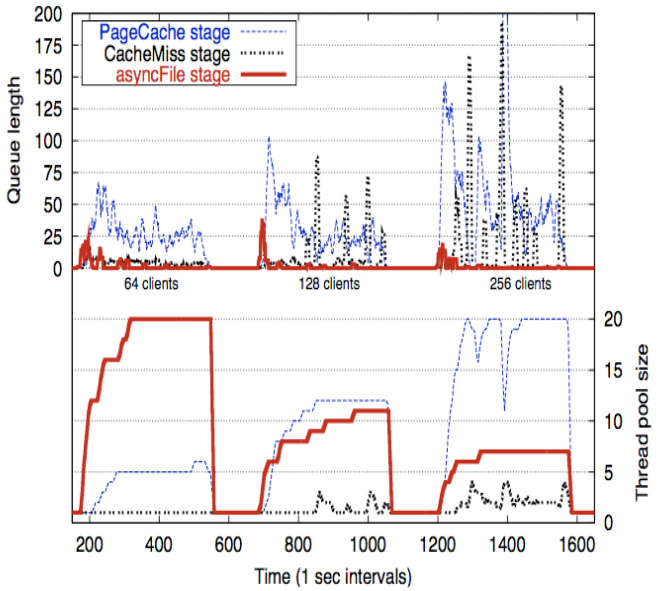
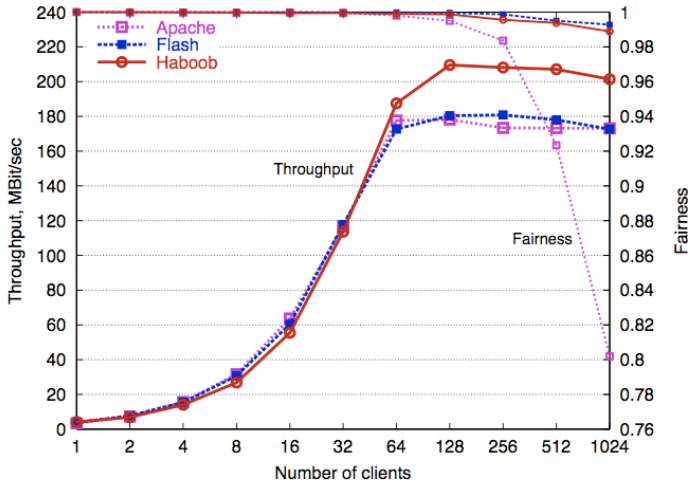


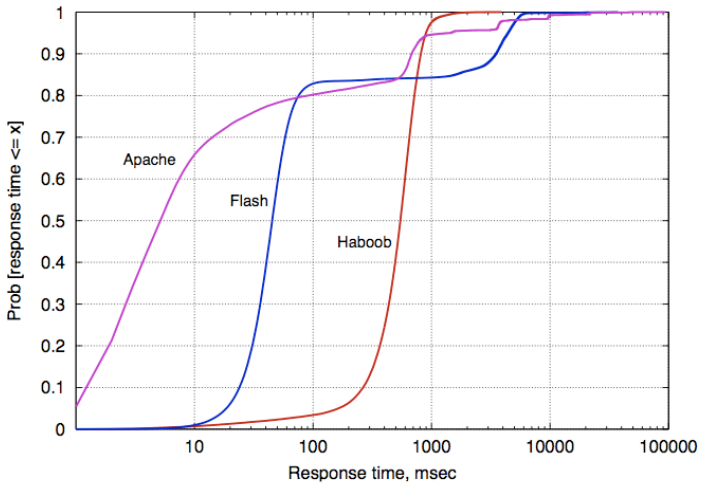
Figure 8: **SEDA thread pool controller:** *This graph shows the operation of the thread pool controller during a run of the Haboob Web server, described in Section 5.1. The controller adjusts the size of each stage's thread pool based on the length of the corresponding event queue. In this run, the queue length was sampled every 2 seconds and a thread was added to the pool if the queue exceeded 100 entries (with a maximum per-stage limit of 20 threads). Threads are removed from the pool when they are idle for more than 5 seconds. The* asyncFile *stage uses a controller threshold of 10 queue entries to exaggerate the controller's behavior.*

Managing concurrency works

- Avoid refusing connections
- Slow down some to bound other responses
  - Note median response time is 10x & 100x slower



(a) Throughput vs. number of clients



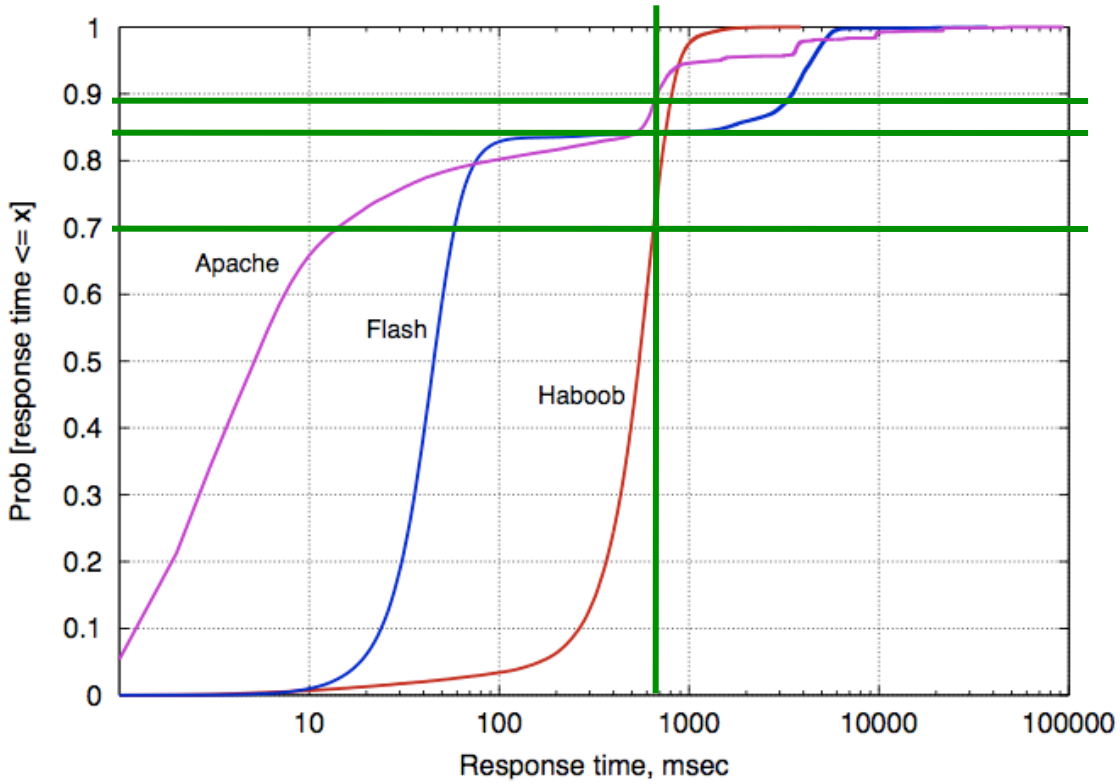
(b) Cumulative distribution of response time for 1024 clients

	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

Figure 12: **Haboob Web server performance:** *This figure shows the performance of the Haboob Web server compared to Apache and Flash. (a) shows the throughput of each server using a fileset of 3.31 GBytes as the number of clients increases from 1 to 1024. Also shown is the Jain fairness index delivered by each server. A fairness index of 1 indicates that the server is equally fair to all clients; smaller values indicate less fairness. (b) shows the cumulative distribution function of response times for 1024 clients. While Apache and Flash exhibit a high frequency of low response times, there is a heavy tail, with the maximum response time corresponding to several minutes.*

Yahoo: >.7s is useless

- That is, lower worst case is not big win
  - Lowers thruput of Haboob by 10-20%
  - Scale all RT %tiles up – Hadoop loses everywhere



(b) Cumulative distribution of response time for 1024 clients

Evaluation

- Excellent, to be expected in 2000+
  - Public systems compared to
  - Full implementations
  - Multiple use cases
  - Lots of numbers
  - A little boring and wordy