SEDA
Staged Event-Driven Architecture

- Problem/Motivations:
  - Internet applications catering to
    - bursty, massively concurrent demands
    - Responsive
    - Robust
    - Fault tolerance
  - Concurrent requests translate to even higher I/O and N/w requests
  - Example:
    - 1.2 billion page views @ Yahoo daily.
    - 10 billion hits @ AOL Web caches daily.

Challenging trends

- Services are becoming complex.
-
- 1. Static content -> Dynamic content
- 2. Deployment issues
- 3. Hosted on general purpose platforms

SEDA – A quick intro.

- Provides a general “framework” for authoring “*highly concurrent*” & “well-conditioned service” instances that “*handle load gracefully*”.
-
- Traditional OS designs versus SEDA design.

SEDA – A quick intro (Contd)

- SEDA combines:
  - Aspects of thread management (Ease of pgmg)
  - Event based programming model (Concurrency)
-
- Using the SEDA framework, applications are developed as “Network of stages”, each with an associated incoming queue.
- Java applications based on SEDA have surprisingly outperformed their C counterparts.

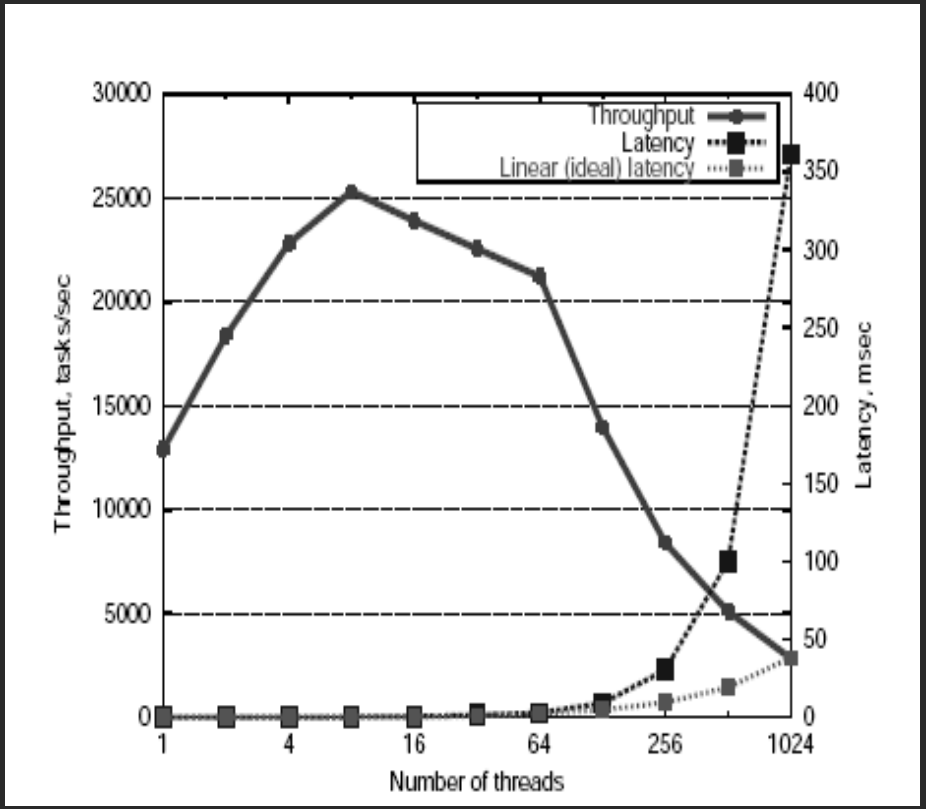
Outline of the rest of the talk:

- Current development frameworks
- SEDA Archictecture
- Haboob Http server
- Gnutella p2p file sharing n/w
- Comparison with other architectures
- Conclusion

Terminology

- Well conditioned service:
  - A service is well conditioned if it behaves like a simple pipeline, where the depth of the pipeline is determined by the path through the network, and the processing stages within the service itself.
  - The key property of a well conditioned service is “graceful degradation”:
    - As the load exceeds capacity, the service maintains high throughput with a linear response-time penalty that impacts all the clients equally, or atleast according to some service specific policy.

Thread based frameworks



A commonly used framework:

- Create a new thread for each request.
- Advantage:
  - Easy to program
- Disadvantages:
  - Overheads for cache/TLB misses, scheduling, lock contentions.

Possible remedies

- Scheduler activations
- Application-specific handlers
- SPIN
- Exokernel
- Etc
- All attempt to give the applications the ability to specialize the policy decisions made by the kernel.
- Bounded thread pools - issues.

Event driven concurrency

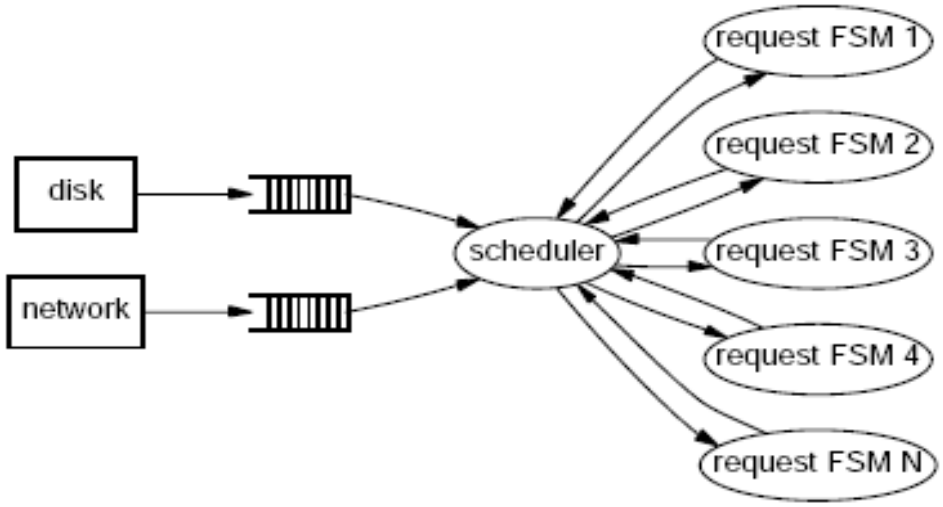
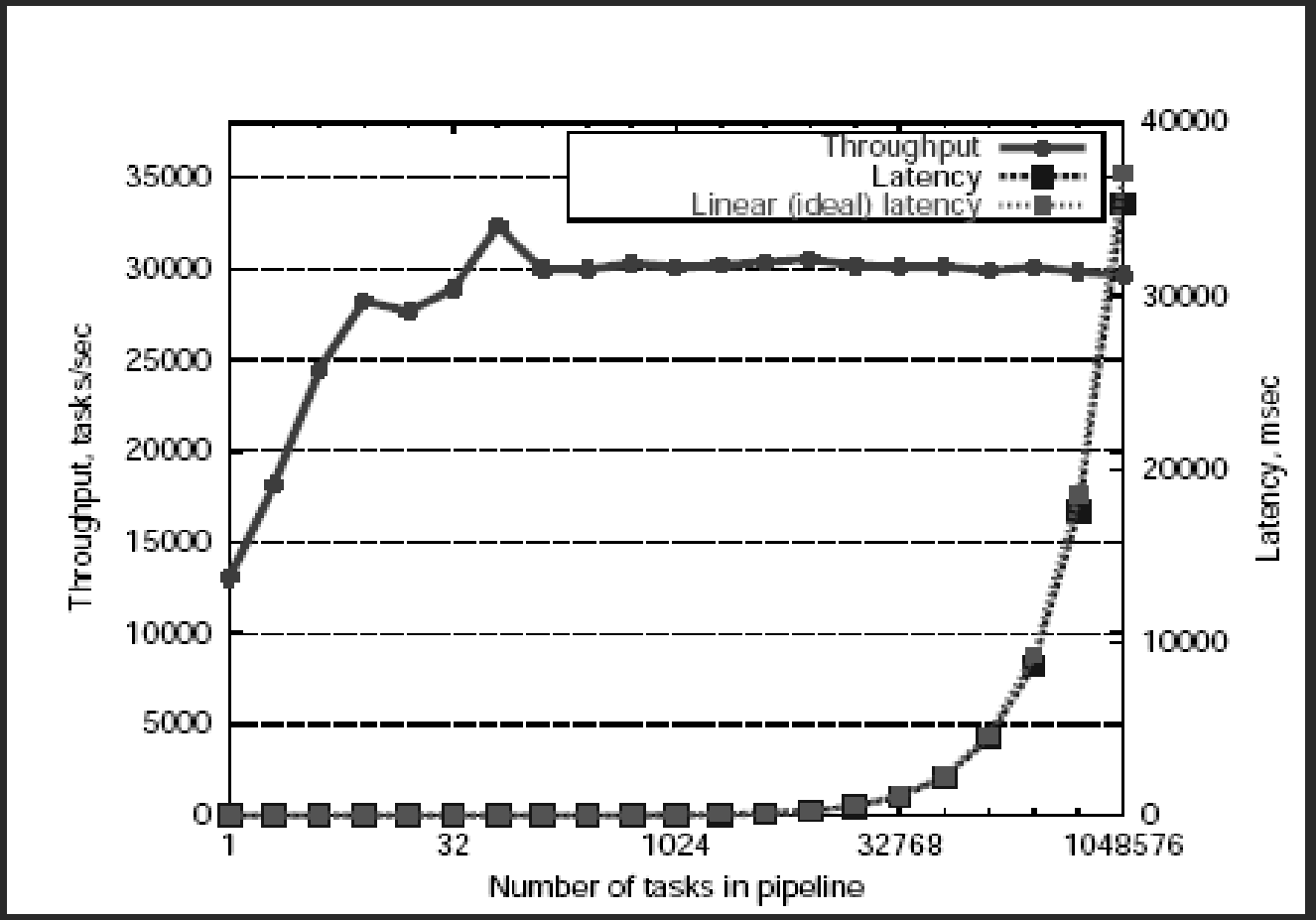


Figure 3: **Event-driven server design:** *This figure shows the flow of events through an event-driven server. The main thread processes incoming events from the network, disk, and other sources, and uses these to drive the execution of many finite state machines. Each FSM represents a single request or flow of execution through the system. The key source of complexity in this design is the event scheduler, which must control the execution of each FSM.*

Event driven mechanisms

- These systems tend to be robust to load, with little degradation in throughput. (Requests vs Events).
- Excess tasks are absorbed in the server's event queue.
- Assumption: Event handling threads do not block. Non-blocking I/O must be employed.

Performance



SEDA Architecture

- Goals:
  - Massive concurrency
  - Well-conditioned service
  - Adapt to changing load conditions
  - Tune resource management
-
- Stage: A fundamental processing unit.
- Has an incoming event queue, a thread pool, and an event handler. Each stage also has a controller for scheduling & thread allocation.

SEDA: Stage

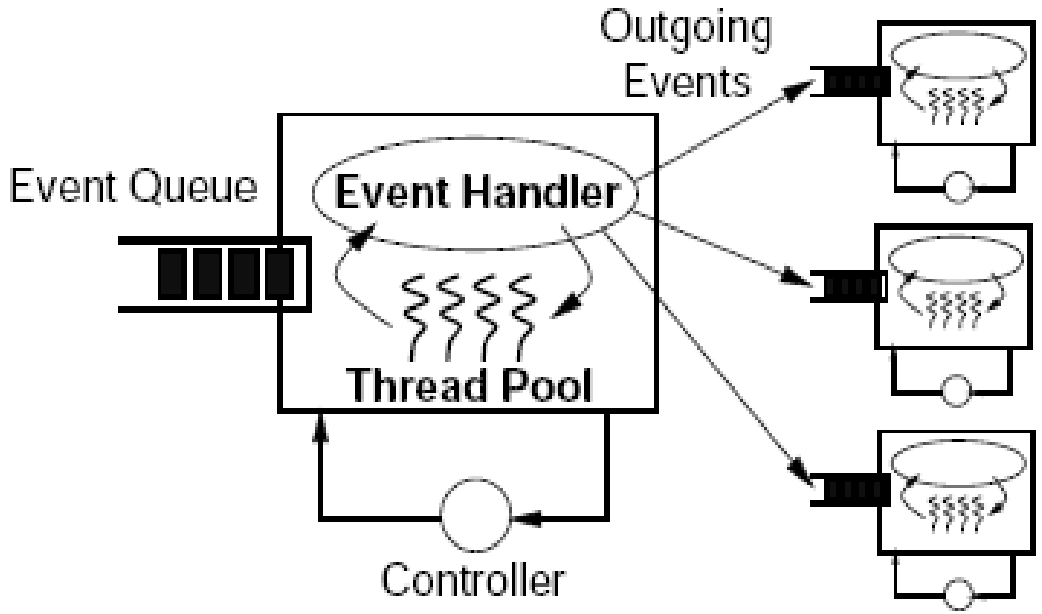


Figure 6: **A SEDA Stage:** *A stage consists of an* incoming event queue*, a* thread pool*, and an application-supplied* event handler*. The stage's operation is managed by the* controller*, which adjusts resource allocations and scheduling dynamically.*

Dynamic Resource Controllers

- Goal:
  - Shield programmers from performance tuning
-
- Resource controllers:
  - Thread pool controller
    - Adjust number of threads in the thread pool
  - SEDA batching controller
    - Adjust number of events processed in each invocation of the event handler.

Sandstorm

- Sandstorm is a SEDA implementation in Java using nonblocking socket i/o (Java NIO library).
- Each application module implements a simple event handler – handleEvents() which processes a batch of events from the incoming queue.
- No worries on thread creation, management.

Sandstorm (Contd)

- Provides Asynchronous network socket layer based on nonblocking I/O provided by the OS
  - AsyncSocket, asyncClientSocket, asyncServerSocket, asyncConnection etc.
  - 3 stages: listen, read, write.
- Provides Asynchronous file I/O layer that uses blocking OS calls and uses threads to expose nonblocking behavior.
  - AsyncFile (provides non-blocking read, write, seek, stat etc)

Applications & Evaluation - Haboob

- A high performance HTTP server
  - Clients issue http requests & wait for responses
  - SPECweb99 benchmark suite used for performance testing.
- Benefits of using SEDA:
  - Constructing Haboob increased the modular design.
  - Each stage provides a robust, reusable component, individually conditioned to load.
  - Test different page cache implementations and file i/o much easily.

Haboob architecture

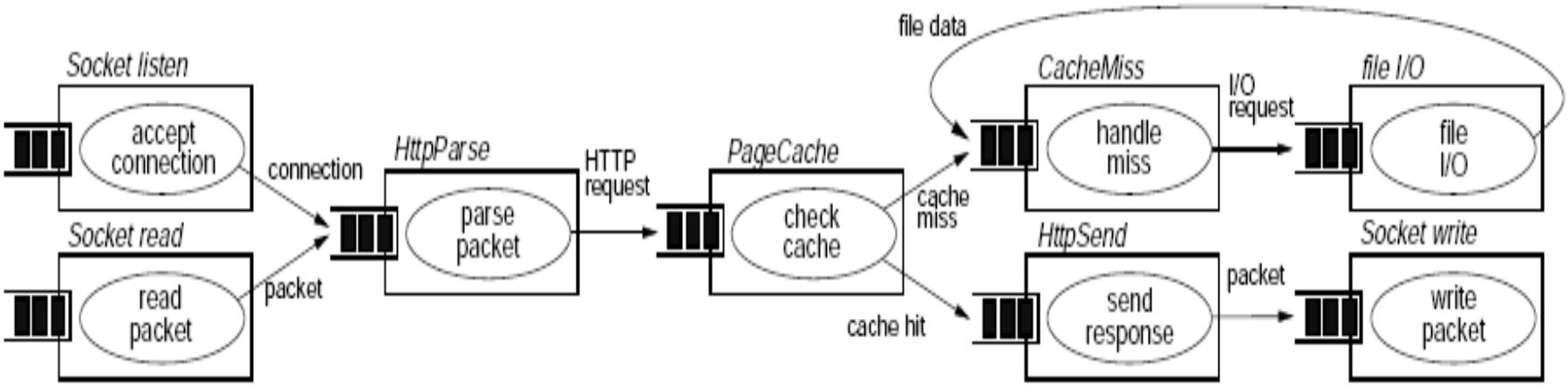
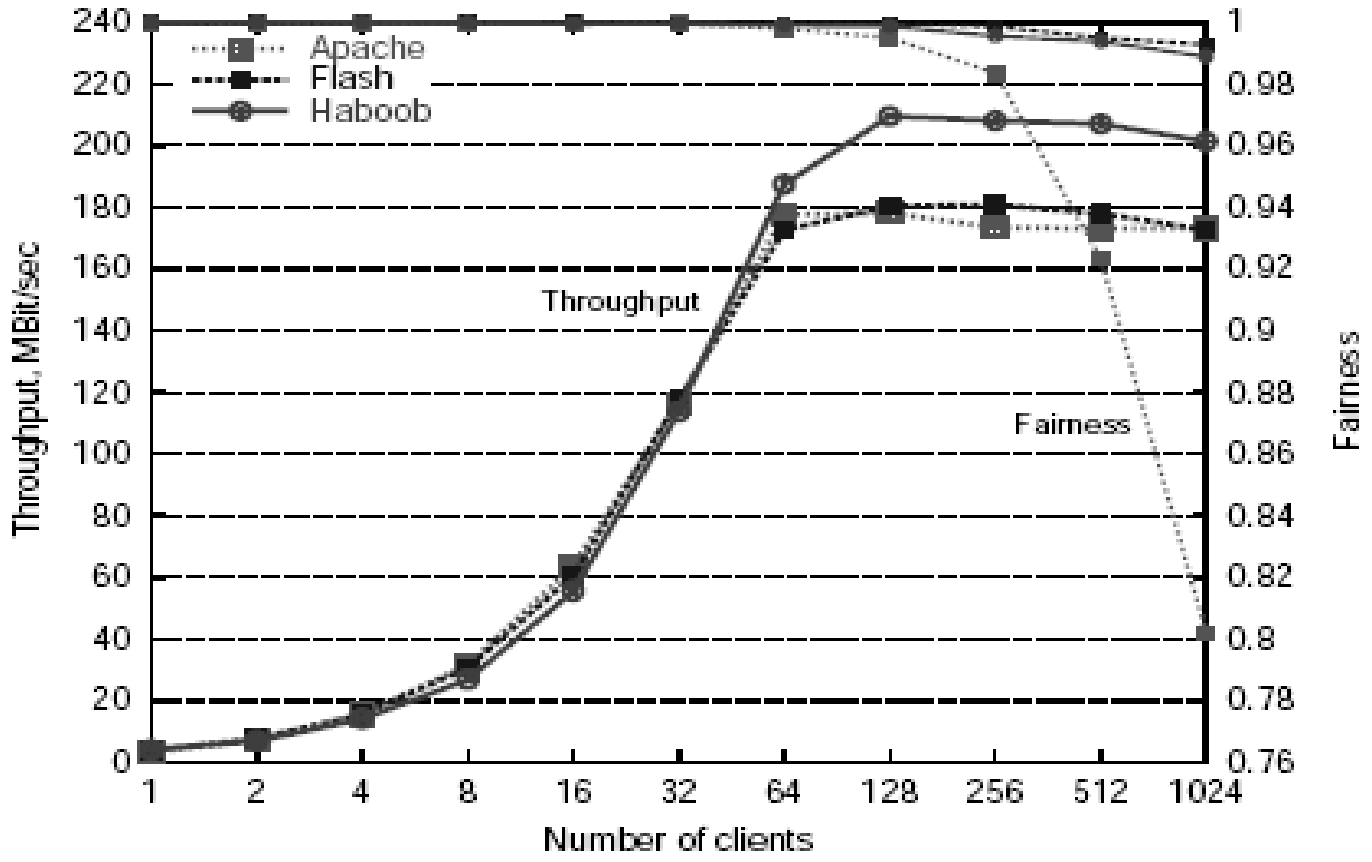


Figure 5: **Staged event-driven (SEDA) HTTP server**: *This is a structural representation of the SEDA-based Web server, described in detail in Section 5.1. The application is composed as a set of* stages *separated by* queues. *Edges represent the flow of events between stages. Each stage can be independently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

- Adaptive load shedding:
  - When overloaded, Haboob adaptively sheds load.
  - The queue threshold is reduced of that particular stage.
  - Example: HttpRecv stage - Error message is returned to the client.

Haboob performance



(a) Throughput vs. number of clients

Gnutella – A Packet router

- A peer to peer file sharing network.
- Search & download files from other peer Gnutella users.
- A node discovers others using a discovery protocol; use ad-hoc multihop routing.
- Architecture has 3 stages:
  - GnutellaServer stage (Connection handling)
  - GnutellaRouter stage (Table, Process, Route)
  - GnutellaCatcher stage (Host discovery)

Gnutella

- Load conditioning policies
  - Threshold incoming queue.
  - Probabilistically drop packets based on queue length.
  - Admit all packets; filter them based on types.

Comparing SEDA with others.

- Recent studies indicate SEDA to perform poorly compared to threaded/event-based systems in C.
- Matt Welsh, counters it:
  - Sandstorm implementation was on Linux 2.2, JDK 1.3
    - Studies have used different environments, JVM's.
  - SEDA n/w layer dependent on several parameters.
    - Tuning these parameters can improve the performance.
  - Goal was to show SEDA had “acceptable performance”, while providing good load

Conclusion

- Measurement & Control is the key, as opposed to fixed resource allocation.
-
- Challenges:
  - Detecting overload conditions
  - Strategy to counter overload
-
- Maybe use SEDA as a new direction in OS design.