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

by, Matt Welsh, David Culler, and Eric Brewer
Computer Science Division
University of California, Berkeley

presented by, Jarett Creason
January 30th, 2008
SEDAR – Staged Event-Driven Architecture

- A robust, high-performance platform for massively concurrent Internet services
- An architecture that can be applied to many different types of services
  - Haboob - Web Server
  - Gnutella – Router
  - Arashi – Dynamic Email Service (not discussed here)
- Designed to be flexible to handle large amounts of load efficiently and effectively
Problem Proposed to Solve

- Give fast, reliable access to millions of users
- Huge variations in server service load
- More and more dynamic content
- Service logic changes often, which requires redesign of algorithms
- Services hosted on general purpose facilities, not specialized hardware
<table>
<thead>
<tr>
<th>Traditional</th>
<th>SEDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes &amp; Threads</td>
<td>Events mixed with threads</td>
</tr>
<tr>
<td>Transparent resource virtualization</td>
<td>Allow applications to make informed</td>
</tr>
<tr>
<td></td>
<td>scheduling decisions</td>
</tr>
<tr>
<td>Very specialized for specific roles</td>
<td>General purpose mechanisms</td>
</tr>
<tr>
<td>Excessive engineering burden to</td>
<td>Easier for programmers to adapt and use</td>
</tr>
<tr>
<td>convert to different architectures</td>
<td>for other applications</td>
</tr>
</tbody>
</table>
Traditional – More in depth

- Thread per request
  - Most common strategy and easy to program
  - High overhead and serious performance degradation
- Virtualization hides presence of limited resources
Traditional – More in depth

- Bounded Thread Pool
  - Limits the total amount of threads possible
  - Solution to serious performance degradation
  - Apache, IIS, IBM WebSphere, etc.
  - Unfairness to clients – requests can queue up for very long periods of time under heavy load
Traditional – More in depth

- Event-Driven Concurrency
  - Small number of threads with a continuous loop processing events from a queue
  - Robust when dealing with heavy loads; little degradation to throughput
  - Scheduling and ordering must be defined, generally very specific and when new functionality is added, the algorithm must be redefined
SEDAR – Goals

- Support Massive Concurrency
  - Avoid performance degradation, make use of event driven execution

- Simplify Construction
  - Shield programmers from the details of scheduling and resource management; support modular construction; support debugging

- Enable Introspection
  - Applications should be able to modify how they behave under heavy load, reschedule and prioritize

- Support Self-Tuning Resource Management
  - Systems should be able to adapt to changing circumstances
SEDAC – Stages

- Self contained application components
  - Includes: event handler, incoming event queue and thread pool
- Managed by a “controller”
- Each stage has a dynamically bounded thread pool
Event queues connect the separate stages

- Event queues can be finite
  - Backpressure – blocking on a full queue
  - Load shedding – dropping events
  - Other – send error to user, or provide degraded service

This provides modularity and more control, but with that comes latency

- Debugger can trace a bogged down stage easily
Goal is to shield programmers from performance tuning.

Automatically adapts resource usage for the stage based on performance and demand.
Sandstorm – SEDA Prototype

- Internet Services Platform
- Completely written in Java
- API for all SEDA functions
  - create and destroy stages
  - queue operations
  - profiling and debugging
- Thread manager
  - handles a pool of threads
Asynchronous I/O Primitives

- Highly concurrent programs need non-blocking I/O so the rest of the program can proceed without having to wait for the slow I/O
- Two SEDA Sandstorm stages for I/O
  - Socket I/O
  - File I/O
Asynchronous Socket I/O

- Easy to use non-blocking socket I/O
  - Provided by the OS
  - Applications create instances of the classes: asyncClientSocket, asyncServerSocket, asyncConnection
- Simply a "SEDA way" of communicating efficiently with sockets
Asynchronous File I/O

- Very different than Sandstorms socket I/O
  - The OS does not provide a way to do asynchronous file I/O (Linux kernel 2.2)
- Uses a thread pool to simulate asynchronous file I/O
"Haboobs" are huge desert dust storms

Made up of 10 total stages

- Four devoted to asynchronous I/O
- HttpParse – accepts new client connections and processes packets
- HttpRecv – accepts HTTP connections and request events
- PageCache – in memory web page cache
- CacheMiss – handles page cache misses
- HttpSend – sends responses to clients
Haboob – Performance

- Trade off between low average response time and low variance in response time

(a) Throughput vs. number of clients

(b) Cumulative distribution of response time for 1024 clients

<table>
<thead>
<tr>
<th>Server</th>
<th>Throughput</th>
<th>RT mean</th>
<th>RT max</th>
<th>Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>173.36 Mbps</td>
<td>143.91 ms</td>
<td>27953 ms</td>
<td>0.98</td>
</tr>
<tr>
<td>Flash</td>
<td>180.83 Mbps</td>
<td>141.39 ms</td>
<td>10803 ms</td>
<td>0.99</td>
</tr>
<tr>
<td>Haboob</td>
<td>208.09 Mbps</td>
<td>112.44 ms</td>
<td>1220 ms</td>
<td>0.99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server</th>
<th>Throughput</th>
<th>RT mean</th>
<th>RT max</th>
<th>Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>173.09 Mbps</td>
<td>475.47 ms</td>
<td>93691 ms</td>
<td>0.80</td>
</tr>
<tr>
<td>Flash</td>
<td>172.65 Mbps</td>
<td>665.32 ms</td>
<td>37388 ms</td>
<td>0.99</td>
</tr>
<tr>
<td>Haboob</td>
<td>201.42 Mbps</td>
<td>547.23 ms</td>
<td>3886 ms</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Gnutella is a p2p file sharing service

3 stages

- GnutellaServer – accepts TCP connections and processes packets
- GnutellaRouter – performs packet routing and routing table maintenance
- GnutellaCatcher – helper stage; used to connect to a site with a list of hosts

Used the profiling and debugging features in Sandstorm to find a memory leak which was causing the program to crash
Conclusions

- SEDA attempts to be an architecture for massively concurrent internet services which are robust, easy to program and handle great variations in load gracefully.
- The use of event queues and dynamic resource control can be used in the future for novel scheduling techniques.
- Was originally implemented with JDK 1.3 and Linux kernel 2.2 (both of which have advanced).
- Current status of SEDA
  - Research has stopped, last release on SourceForge July 2002
  - “The most fundamental aspect of the SEDA architecture is the programming model that supports stage-level backpressure and load management. Our goal was never to show that SEDA outperforms other server designs, but rather that acceptable performance can be achieved while providing a disciplined approach to overload management. Clearly more work is needed to show that this goal can be met in a range of application and server environments.”
  - http://www.eecs.harvard.edu/~mdw/proj/seda/