Efficient Network Polling with Fine-Grained Interval Control

Eiji Kawai, Youki Kadobayashi, and Suguru Yamaguchi
Nara Institute of Science and Technology
JAPAN
Background

- Busy network server handles a huge number of concurrent connections
  - Increasing network bandwidth
  - Over 100Mbps
  - Sophisticated connection handling
  - HTTP persistent connection

- Polling I/O is used on traditional Unix
  - select() / poll()
  - Scalability issue
Related Work

- New Event Notification Mechanism
  - OS notifies server process of events on the interest set of connections [Banga99]
  - kqueue (FreeBSD) [Lemon01]
  - /dev/poll (Solaris, Linux) [Provos00]

- POSIX real-time signals
  - Server process retrieves events from the signal queue
    - More than one signal at one retrieval [Provos01]
    - Queue overflow problem [Chandra01]

**Problem:** Modifications in programming model and operating system are required
Goal

- Improving the scalability of polling I/O
  - Without major modification in its programming model
  - Without major modification in the underlying operating system
Polling I/O Programming Model

1. Create Socket List
2. Invoke select() or poll()
3. Process I/O-Ready Sockets
Scalability Issue of Polling I/O

- Well-known problem: Processing cost
  - Scan a large list of sockets

- Real problem: Too frequent invocation
  - Low efficiency
    - Efficiency = output / input
    - Short invocation interval means small output
  - Processor cycle starvation
Interval Control

- Control the invocation interval to avoid too frequent invocation

Create Socket List

**Interval Control with Self-Blocking**

Invoke select() / poll()

Process I/O-ready sockets
Performance Evaluation

- Benchmarking with Web Polygraph
  - Implement the interval control mechanism in a web server accelerator (reverse proxy server)
  - Two cases
    - Persistent connections disabled
      - Small number of concurrent connections (~200)
    - Persistent connections enabled
      - Large number of concurrent connections (~7000)
  - Service latency / throughput
Service Delay
(persistent connections disabled)

Response Time (All)
(without persistent connections)

- poll() interval = 0 ms
- poll() interval = 5 ms
- poll() interval = 10 ms
- poll() interval = 15 ms
- poll() interval = 20 ms

Request Rate (req/s)

Response Time (ms)
Service Delay
(persistent connections enabled)

Response Time: All
(with persistent connections)

poll() interval = 0 ms
poll() interval = 10 ms
poll() interval = 20 ms
poll() interval = 30 ms
poll() interval = 40 ms

Response Time (ms)
Request Rate (req/s)
Kernel Profiling

- kernprof: Linux kernel profiler from SGI
  - CPU usage statistics (PC sampling)
  - Total, Kernel, poll()
Processor Usage of poll() in Kernel (persistent connections disabled)

CPU Usage: poll() in Kernel (without persistent connections)

- poll() interval = 0 ms
- poll() interval = 5 ms
- poll() interval = 10 ms
- poll() interval = 15 ms
- poll() interval = 20 ms

Request Rate (req/s)

CPU Usage (%)

Nov. 17, 2003
Processor Usage of poll() in Kernel (persistent connections enabled)

CPU Usage: poll() in Kernel (with persistent connections)

- poll() interval = 0 ms
- poll() interval = 10 ms
- poll() interval = 20 ms
- poll() interval = 30 ms
- poll() interval = 40 ms
Conclusion

- Interval control on polling I/O greatly improves server performance
  - Small effect on processor usage behavior in case with persistent connections
  - Improvement on processor usage is our future work
End

Thank you for your attention.