Proportional-Share Scheduling of Operating System Services for Real-Time Applications

Kevin Jeffay
F. Donelson Smith
Arun Moorthy
James H. Anderson

Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/Research/Dirt/

Motivating Problem

- Real-time and non-real-time tasks present in current day workload
- Aim: To support this workload on general purpose desktop computers

Potential Solutions/Related Work

- Build a new real-time operating system
  » Rialto (Jones et al., 1997)
- Real-time extensions to existing operating systems
  » Real-Time Mach (Tokuda et al., 1990)
  » SMART Solaris System (Nieh et al., 1997)
- Virtual Machine Emulation
  » Real-Time Linux (Barbarnov & Yodaiken)
  » Real-Time IBM Microkernel (Bollella & Jeffay, 1995)

Proportional Share Scheduling of User Processes

- Proportional share ensures *fair* allocation
- Processes assigned a weight $w$
  » weight determines process's *share*
- Fairness can be used to ensure real-time execution
  » Real-time processes have a fixed share
  » Non-real-time processes have a variable share

\[
\sum w_i = 6 \quad \sum w_i = 13.5
\]

\[
w_1 = 3, \quad w_2 = 4.5, \quad w_3 = 6
\]

\[
w_1 = 2, \quad w_2 = 1, \quad w_3 = 1
\]
Proportional-Share-based Real-Time Extensions

- SFQ SVR4 Unix (Goyal et al., 1996)
- Mach- and FreeBSD-based Lottery Scheduling (Waldspurger & Weihl, 1994)
- FreeBSD: EEVDF version (Stoica et al., 1996)

All perform Proportional Share Scheduling at the User-Level

Scheduling of OS Services
Integrated Resource Allocation

- Extensive research
- Sophisticated process scheduling
- Relatively less attention
- Process-independent scheduling

Undesirable Effect: Improper Allocation of Resources within the Kernel might adversely affect Real-Time performance

Solution: Integrate Scheduling of Operating System Activities and Application Scheduling

Example: Protocol Processing in BSD Unix

+ Advantages
  » Fast response
  » High throughput
– Disadvantages
  » Static priority network processing
  » Receive livelock
  » No packet distinction

Schedule Protocol Processing exactly like any other activity

Example: Real-Time Mach (Lee et al., 1996)
- Protocol stack is a library
- Protocol processing is
  » schedulable
  » fully preemptible

Real-Time Network & Protocol Processing Principles
Proportional-Share Network & Protocol Processing

Logical Process for Protocol Processing

Explicitly Schedule Protocol Processing

App 1 App 2 App 3

App 1 App 2 App 3

socket

socket

socket

Protocol Processing Destination Queues

- One queue per socket
  → as in LRP

- Varying queue lengths
  → Queue length is a function of process weights

Real-Time Network & Protocol Processing Principles (Contd.)

Early Packet Demultiplexing

Example: Lazy Receiver Processing (Druschel & Banga, 1996)

- One queue per socket on receiver
- Lazy protocol processing

Hierarchical Packet Scheduler

- Assign weight/cost to each packet

- Proportional-share sub-allocation of quantum
### Experimental Setup
(FreeBSD 2.2.2-Release)

- Audio receiver (5% CPU utilization)
- M-JPEG receiver (45% CPU utilization)
- tftp receiver (20% CPU utilization)
- Dhrystone (100 - x% CPU util.)

100 Mbps Ethernet

- Audio Sender 50 packets/sec
- M-JPEG Sender 90 packets/sec
- tftp Sender Normal: 200 packets/sec
  Broken: 1000 packets/sec

### Outline of Experiments

1. Baseline: Unmodified FreeBSD
2. Prop-share at user-level
3. Prop-share at user-level & IP
4. Prop-share at user-level & IP with destination queues
5. Prop-share at user-level & IP with destination queues & packet scheduling

\[
\text{In each trial:} \\
\begin{align*}
\text{• Regular senders} \\
\text{• Bursty senders} \\
\text{• Broken tftp sender}
\end{align*}
\]

### Experimental Results
Unmodified FreeBSD: 1ms quantum

<table>
<thead>
<tr>
<th>App</th>
<th>Audio</th>
<th>M-JPEG</th>
<th>tftp</th>
<th>Dhrystone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal senders</td>
<td>3000 packets</td>
<td>3300 received/5400 packets</td>
<td>12000 packets</td>
<td>7.3 * 10^6 iterations</td>
</tr>
<tr>
<td>Broken tftp sender</td>
<td>3000 packets</td>
<td>2400 received/5400 packets</td>
<td>12000 packets</td>
<td>5.5 * 10^6 iterations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App</th>
<th>Audio</th>
<th>M-JPEG</th>
<th>tftp</th>
<th>Dhrystone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal senders</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
<tr>
<td>Broken tftp sender</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
</tbody>
</table>

### Proportional-Share Scheduling at the User-Level

<table>
<thead>
<tr>
<th>App</th>
<th>Audio</th>
<th>M-JPEG</th>
<th>tftp</th>
<th>Dhrystone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal senders</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
<tr>
<td>Broken tftp sender</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App</th>
<th>Audio</th>
<th>M-JPEG</th>
<th>tftp</th>
<th>Dhrystone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal senders</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
<tr>
<td>Broken tftp sender</td>
<td>3000 packets</td>
<td>5400 packets</td>
<td>12000 packets</td>
<td>4.6 * 10^6 iterations</td>
</tr>
</tbody>
</table>
Proportional-Share at the User-Level and IP

<table>
<thead>
<tr>
<th>Normal senders</th>
<th>Broken tftp sender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>3000 packets</td>
</tr>
<tr>
<td>M-JPEG</td>
<td>5400 packets</td>
</tr>
<tr>
<td>tftp</td>
<td>12000 packets</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>4.9 * 10^6 iterations</td>
</tr>
<tr>
<td></td>
<td>757 received/3000 packets</td>
</tr>
<tr>
<td></td>
<td>2000 received/5400 packets</td>
</tr>
<tr>
<td></td>
<td>12000 packets</td>
</tr>
<tr>
<td></td>
<td>15200 + 32800 drops</td>
</tr>
<tr>
<td></td>
<td>8.8 * 10^6 iterations</td>
</tr>
</tbody>
</table>

Broken tftp sender (No packet scheduling)

<table>
<thead>
<tr>
<th>Broken tftp sender (With packet scheduling)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
</tr>
<tr>
<td>M-JPEG</td>
</tr>
<tr>
<td>tftp</td>
</tr>
<tr>
<td>Dhrystone</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Broken tftp sender (With packet scheduling)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
</tr>
<tr>
<td>M-JPEG</td>
</tr>
<tr>
<td>tftp</td>
</tr>
<tr>
<td>Dhrystone</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Proportional-Share at the User-Level and IP Destination Queues (With and Without Packet Scheduling)

Proportional-Share Scheduling of Operating System Services for Real-Time Applications

Conclusions

- Operating system activities need to be scheduled as well as user processes
- Proportional-share is effective in both domains
- Developed a limited proportional-share version of FreeBSD
  - Network subsystem in kernel is implemented in a prop-share manner
  - User processes are scheduled in a prop-share fashion
  - Solution to the receive livelock problem