FAST-OS

“FAST-OS (Forum to Address Scalable Technology for runtime and Operating Systems) provides a forum for the presentation and discussion of research results related to the development of operating and runtime systems for very large scale systems.”

funding: US government

FAST-OS Workshop serves as the PI meeting for the projects funded by the FAST-OS program and will include progress reports from these projects.

The workshop is open and participation by the OS research community is encouraged.

More info at [http://www.cs.unm.edu/~fastos](http://www.cs.unm.edu/~fastos)
Petascale HPC

World View:
The Road to Petaflops & Beyond

- Login Nodes
- Mgmt Nodes
- Compute Nodes
- I/O Nodes
- Storage Targets

~500K CPUs

Pete Beckman
Focus

- very large scale HPC systems: special purpose lightweight OS or fully featured OS?
  - services, versatility, flexibility lead to noise

- OS noise: OS activity that is unrelated to the application
  - big problem for HPC, large scale parallel jobs
  - occasional delays are magnified because many threads/nodes suffer
  - often leave a CPU “idle” for system tasks on each node (e.g., BG/L)

- the “fatter” the OS the stronger the noise
  - clock interrupts are a major source of noise
  - and other interrupts are not so good, either
A Brief History of Time(-Sharing)

- time-sharing computers
  - computers are expensive, users share resources
  - applications do not run as fast as they could: expansion factor (≈ 10 for typical HPC)
  - SMP, gang scheduling
- clusters
  - a node is dedicated to one application at a time
  - interrupts, daemons, networking
  - get rid of parts of the kernel
    - e.g., networking: Myrinet, Quadrics: apps talk to HW directly
# Noise Scales

## Noise: It’s all about scale...

<table>
<thead>
<tr>
<th>Source</th>
<th>Magnitude</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache miss</td>
<td>100 ns</td>
<td>accessing next row of a C array</td>
</tr>
<tr>
<td>TLB miss</td>
<td>100 ns</td>
<td>accessing infrequently used variable</td>
</tr>
<tr>
<td>HW interrupt</td>
<td>1 µs</td>
<td>network packet arrives</td>
</tr>
<tr>
<td>PTE miss</td>
<td>1 µs</td>
<td>accessing newly allocated memory</td>
</tr>
<tr>
<td>timer update</td>
<td>1 µs</td>
<td>process scheduler runs</td>
</tr>
<tr>
<td>page fault</td>
<td>10 µs</td>
<td>modifying a variable after <code>fork()</code></td>
</tr>
<tr>
<td>swap in</td>
<td>10 ms</td>
<td>accessing load-on-demand data</td>
</tr>
<tr>
<td>pre-emption</td>
<td>10 ms</td>
<td>another process runs</td>
</tr>
</tbody>
</table>

Our definition: only what the user code cannot control is “noise”
ZeptoOS: Small Linux for Big Machines

- research focus:
  - fundamental limits, advanced designs and petaflop scale
  - managing and optimizing OS
  - gathering and studying performance data

- approach: modified Linux on BG/L I/O and compute nodes, specialized I/O daemon (ZOID)

- “don’t attribute to OS noise what can be adequately explained by stupidity” (e.g., do not start `lpd` on an HPC node)

- injecting noise: on a vanilla Linux with minimal modifications an application that does 300 barriers per second will run 0.3% slower on 100K nodes
PROSE (Eric Van Hensbergen)

- Partitioned Reliable Operating System Environment
- application-specific kernels
  - fine-grained control over OS services: scheduling, memory allocation, interrupt handling (or lack thereof)
  - reliability (up to and including formal verifications)
  - HW support: hardware-specific features not suitable for general purpose OS
Virtualization
PROSE Approach
PROSE Approach II

- run applications in dedicated partitions
- execution environment makes starting a partition as easy as starting an application
- development environment allows developing specialized kernels as easily as developing applications (library-OS)
- based on rHype
  - 30K LOC for both x86 and PPC
  - same interfaces as pHype
PROSE Noise
Light- and Right-Weight Kernels

- LWK — get rid of most of the kernel
  - no filesystems, no sockets, no paging or virtual memory, no security, no scheduling...
  - no shared libraries
  - no debugging...
  - does it go too far?

- RWK
  - don’t take for granted that LWK are necessary
  - try to trim down Linux and/or use Plan9 to compare
  - develop metrics
Right-Weight Kernels

- What is light-weight?
  - We probably don’t need a printer daemon
  - Do we need a filesystem? Maybe...
  - What ro remove?
  - How to benchmark?

- What is right-weight?
  - LWK on BG/L c-nodes: 80 system calls
  - Linux: 300 system calls
  - Plan9: 40 system calls
  - But Plan9 has clock interrupts

- Develop simulators, measure
difficulty in achieving balanced partitioning and dynamic scheduling on very large machines can limit scaling
today application programmers often manage resources explicitly
goal: delegate these functions to a parallel OS
system management (scheduling, event notification, job management) does not scale
goal: OS needs to be aware of the requirements of parallel applications
directions:
full featured Linux on Blue Gene
parallel aware scheduling
virtualization: fault tolerance, resource management
Virtualization on Blue Gene

- divide the computation into a large number of pieces, independent of the number of processes
- let the runtime system map the pieces to processors
- e.g., Adaptive MPI (AMPI)
- implement MPI processes as user-level migratable threads

- load balancing:
  - centralized — not scalable
  - distributed — slow balancing
  - hybrid — group nodes hierarchically, within each group use centralized scheme
  - topology aware
Checkpointing, Fault Detection, Restart

- migrate when a fault is imminent

- if one CPU fails the other 99,999 should not scurry to checkpoints
  - everyone logs messages
  - asynchronous checkpoints

- when one CPU fails and restarts, the failed objects restart from their checkpoints, their acquaintances resend messages, the restarted objects catch up

- but sooner or later we will stall, because we need to wait for the crashed processor
  - migrate work from restarted processors to waiting ones
Future: FAST-OS 2 Topics

- resource management: local/global, multicore, heterogeneous, power
- system management: usage models, scalability, RAS
- virtualization for HPC
- adaptability of OS/runtime to application needs
- performance measurements
- scalable fault handling, communicating fault information
- OS noise/interference: design, hardware support, mitigation strategies
- interfaces to allow applications query hardware characteristics, communication resources