**STORM: Scalable T0ol for Resource Management**

**Goals**

Scalable, lightweight and for resource management:
- Allocation of processes to processors
- Graceful distribution of executable and data files
- Job launching
- Coordinated process context switching
- Increase the usability of a cluster
- Checkpointing and fault-tolerance (work in progress)

**Resource Management**

For more information:
- “STORM: Lightning Fast Resource Management”
- In Proceedings of the BIG/NOW/SCED Conference
- Albuquerque, November 26-28, 2002
- http://www.c3.lanl.gov/~fabrizio

Talk
- Wednesday, November 20 at 4:30 p.m.

**Job Launching**

- I/O-bypass mechanism to transmit the files directly from NIC to disk
- Hardware collective communication to distribute program binaries and data files
- Allocation of processes to processors
- Global distribution of executable and data files

**For sequential OS**

- Typical range for sequential OS
- Context-switching overhead is negligible for time quanta of 2ms or more. In fact, Figure 3 shows that context switching an entire parallel job can be prohibitive. STORM's global context switch is so fast that gang scheduling is practical, enabling interactive applications such as visualization and computational steering.

**Table 1**

<table>
<thead>
<tr>
<th>Resource Manager</th>
<th>Minimal feasible quantum (ms) on 64 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORM</td>
<td>100 ms</td>
</tr>
<tr>
<td>RMS</td>
<td>30,000 ms</td>
</tr>
</tbody>
</table>

**Figure 2**

- Shows the measured and predicted launch times of a 12MB executable launch time, which remains well under a second even for 16K nodes.

**Figure 3**

- Shows the effect of using different time-quanta values when running an application kernel in production use at LANL. Context-switching overhead is currently not widely used on production systems because the overhead is negligible for time quanta of 2ms or more. In fact, Figure 3 shows that context switching an entire parallel job can be prohibitive. STORM's global context switch is so fast that gang scheduling is practical, enabling interactive applications such as visualization and computational steering.

**Figure 4**

- Shows measured results for job launching on a 2MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 5**

- Shows measured results for job launching on a 8MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 6**

- Shows measured results for job launching on a 12MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 7**

- Shows measured results for job launching on a 4MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 8**

- Shows measured results for job launching on a 8MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 9**

- Shows measured results for job launching on a 12MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 10**

- Shows measured results for job launching on a 4MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 11**

- Shows measured results for job launching on a 8MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 12**

- Shows measured results for job launching on a 12MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.

**Figure 13**

- Shows measured results for job launching on a 4MB executable without CPU intervention. I/O-bypass mechanism to transmit the files directly from NIC to disk.