Xen Performance Monitoring

Team: Rob Gardner (HP Labs)
Lucy Cherkasova (HP Labs)
Diwaker Gupta (UCSD)
Motivation

• Open question: which CPU scheduling and resource allocation policies provide best service in a given circumstance?

• We need a performance tool capable of enabling a thorough evaluation of different CPU scheduling algorithms
XenMon

- A light-weight, non-intrusive monitoring system that accounts for:
  - CPU usage by different guest VMs;
  - Waiting time: amount of time a domain spends waiting run on the CPU;
  - Blocked time: the amount of time a domain spends blocked (or sleeping);
  - I/O count: the number of memory page exchanges between Dom0 (driver domain) and guest domains. This metric helps to understand the I/O communication patterns;
  - Execution count: the number of times a domain was scheduled to run.

- These metrics are reported at three time scales: execution periods, 1 sec, 10 sec.
XenMon in detail

- The XenMon infrastructure consists of various components:
  - *Xen Trace Facility* in the hypervisor;
  - *Additional QoS Trace calls* in the hypervisor;
  - *xenbaked*: a user process that takes raw event data and accumulates it into a more usable form;
  - *xenmon*: Python-based program that processes accumulated data into periodic metrics and displays them or logs them;
  - *Potential future interfaces*: Network access (via RPC) to processed data, etc.
XenMon Architecture

- Dom-0
  - Other frontends
  - XenMon frontend
  - Xenbaked: "cook" events
  - Xentrace: catch events
  - Xentrace: generate events

- VM

Xen

January 18, 2006
Xen Trace Facility

The TRACE procedure/macro is called at any time to create a new trace record.

…

xen_do_something_cool(stuff);
TRACE(COOL_EVENT, dom_id);

…
Anatomy of a Trace Record

Each trace record contains:

- Event Identifier
- CPU number where code is executing
- Time stamp
- Optional parameters

\[ \text{TRACE(TRAUC\_AD\_SWI\_CH\_INF\_NEXT, domid, wait\_time, alloc\_time)} \]

Means that \text{domid} is about to start running; It had to wait on the run queue for \text{wait\_time} and is scheduled to run for \text{alloc\_time}.
Our Trace Events

- TRC_SCHED_SWITCH_INFNEXT: some domain just started running
- TRC_SCHED_SWITCH-INFPREV: some domain just stopped running
- TRC_SCHED_SLEEP: some domain blocked
- TRC_SCHED_WAKE: some domain unblocked
- TRC_MEM_PAGE_FLIP: memory page exchange happened
- A few other infrequently used ones for domain creating, destruction, etc.
xenbaked

- Background user process that “cooks” the raw trace data into an easily digestible form
- For instance, domain blocked time can be computed by observing a sequence of trace events:
  - SLEEP event for dom X
  - arbitrary sequence of other events
  - WAKE event for dom X
Periodic Accumulation

• Xenbaked accumulates each metric every $\frac{1}{10}$th second (configurable via command line option)

• So if a domain is blocked for $\frac{1}{2}$ second, for example, we have a sequence of blocked times:

• $\ldots 0, 0, 0, 100\text{ms}, 100, 100, 100, 100, 0, 0, \ldots$

• Each number represents time that the domain was blocked during a $\frac{1}{10}$th second period
Periodic Accumulation

• Enough data is kept to create a 10 second history
• Periodic accumulation is done for a number of metrics:
  − CPU time gotten
  − Time blocked
  − Time waiting
  − Time allocated
  − Execution count
  − I/O count
Periodic Accumulation

- Picture a circular buffer of 100 data points for each of the metrics:

- Array Datapoints[100] of {
  - Cpu time gotten
  - Time blocked
  - Time waiting
  - Time allocated
  - Execution count
  - I/O count
}

January 18, 2006
Periodic Accumulation

• Now duplicate all that for each domain:

• Array Domains[NDOMAIN] of {
  • Array Datapoints[100] of {
    • CPU time gotten
    • Time blocked
    • Time waiting
    • Time allocated
    • Execution count
    • I/O count
  }
}
Periodic Accumulation

• And duplicate again for each CPU:

• Array CPUs[ncpu] of {
  • Array Domains[NDOMAIN] of {
    • Array Datapoints[100] of {
      • CPU time gotten
      • Time blocked
      • Time waiting
      • Time allocated
      • Execution count
      • I/O count
    }
  }
}
Data Sharing

• That huge data structure is kept in a shared memory segment
• This makes it accessible to other applications
• For instance, an app could be written to display the data...
XenMon Screen Shot

<table>
<thead>
<tr>
<th></th>
<th>Last 10 seconds</th>
<th>Last 1 second</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>860.32 ms</td>
<td>822.00 ms</td>
</tr>
<tr>
<td>1</td>
<td>94.04 ms</td>
<td>47.97 ms</td>
</tr>
<tr>
<td>0</td>
<td>116.39 ms</td>
<td>177.90 ms</td>
</tr>
<tr>
<td>0</td>
<td>23957/s</td>
<td>5806</td>
</tr>
<tr>
<td>1</td>
<td>94.84 ms</td>
<td>7.72%</td>
</tr>
<tr>
<td>1</td>
<td>1.01 s</td>
<td>5.00 ms</td>
</tr>
<tr>
<td>1</td>
<td>22.97 ms</td>
<td>0.00 ms</td>
</tr>
<tr>
<td>2</td>
<td>17570/s</td>
<td>21.26 ms</td>
</tr>
<tr>
<td>2</td>
<td>171.35 ms</td>
<td>104.82 ms</td>
</tr>
<tr>
<td>2</td>
<td>753.20 ms</td>
<td>117.95 ms</td>
</tr>
<tr>
<td>2</td>
<td>193.62 ms</td>
<td>581.12 ms</td>
</tr>
<tr>
<td>2</td>
<td>5397/s</td>
<td>18.44</td>
</tr>
<tr>
<td>7</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
</tr>
<tr>
<td>7</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
</tr>
<tr>
<td>7</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
</tr>
<tr>
<td>7</td>
<td>0.00 ms</td>
<td>0.00 ms</td>
</tr>
</tbody>
</table>

Records lost: 302 (Min: 0, Max: 16)
Records lost: 62 (Min: 0, Max: 16)
XenMon Status

- Released to the Xen community
- Accepted into Xen 3.0
- Runs on x86, x86-64 and Itanium (IA64)
- Performance case study using XenMon is available as HP Labs Report HPL-2005-187
- Future Enhancements
  - A more reliable trace buffer facility with flow control
  - More metrics
  - Optimized display script – current python implementation sucks up some cpu time
  - Graphical display front-end