I/O Virtualization Bottlenecks in Cloud Computing Today

Jeffrey Shafer – Rice University
Cloud Computing

• Hot topic in popular media

• Builds upon established (trusted?) virtualization technology
  • Install applications not onto native machine, but into virtual machine images
  • Treat datacenter as a generic computing resource
    • Start / stop / migrate application images on demand

• Take the next step with cloud computing
  • Out-source the generic datacenter
    • Let someone else manage it
    • Pay only for what you use
  • Pioneered by Amazon Elastic Compute Cloud (EC2)
    • “Platform as a service” abstraction
Private Cloud Computing

• What if I’m not ready to trust the cloud?
  • Security concerns
    • Who has access to my data?
  • Performance / quality of service concerns
    • How many other customers are sharing the same server, network, or storage array?
  • Vendor lock-in
    • What if Amazon raises prices?

• Private cloud computing
  • Build your own cloud “behind the firewall”
Eucalyptus

- Allows creation of private clouds
- Open-source cloud computing framework
  - Linux-centric: Many distributions, KVM or Xen virtualization
- API compatible with Amazon platform
  - Allows re-use of common administrative tools
  - Allows private clouds to burst to public clouds if desired
- Ubuntu Enterprise Cloud – New in version 9.10
  - Pre-packaged Eucalyptus installation
  - 30 minute “cloud-in-a-box”
Today’s Talk – Data-Intensive Computing

• How well does this cloud computing platform run data-intensive applications?
  • Hadoop – open-source MapReduce framework written in Java
    • Convenient way to parallelize computation across a cluster
    • Target applications: web indexing, data mining, log file analysis, machine learning, scientific simulation, etc...
    • Commonly run in a cloud environment by those who can’t afford a dedicated cluster (or don’t need one full-time)
      • Equivalent to Amazon Elastic MapReduce product

• Summarize out-of-the-box performance and configuration options

• Discuss ways to increase performance
Eucalyptus Architecture (Simplified)

- Cloud Controller (CLC)
  - High-level management
- Cluster Controller (CC)
  - Gateway to cluster nodes
- Elastic Block Storage (EBS)
  - Block-based network disks
- Node Controller (NC)
  - Manages VM instances
Data-Intensive Computing Performance

- Hadoop MapReduce framework
  - 10GB read/write tests (streaming sequential access)
  - Used local storage (disk attached to same node)
  - Measured execution time (seconds)

<table>
<thead>
<tr>
<th>Environment</th>
<th>Write (s)</th>
<th>Read (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Virtualized</td>
<td>113</td>
<td>116</td>
</tr>
<tr>
<td>Virtualized</td>
<td>6826</td>
<td>196</td>
</tr>
</tbody>
</table>

- Virtualized system not CPU limited (> 90% idle)
  - Storage bandwidth the bottleneck?
Eucalyptus Storage Options

- **Choice #1: Local storage**
  - File on local disk mapped into guest domain
  - Equivalent to Amazon local instance storage
Eucalyptus Storage

- **Choice #2: Network storage**
  - File on remote disk (on storage server)
  - Network disk server exports file across the network
    - ATA over Ethernet protocol
    - Lightweight encapsulation of ATA requests, non-routable
  - Host domain runs device driver to access network storage
  - Abstraction: To clients, storage is still local
  - Equivalent to Amazon Elastic Block Storage (EBS)
Experiment Setup

- Ubuntu Enterprise Cloud with KVM
- 500 GB Seagate SATA hard drives
- 1 guest per machine – Want to be disk-bound, not compute-bound
- 1 disk per guest – Data-intensive applications “share” poorly
Eucalyptus Storage Performance

- DD synthetic test
  - Same access pattern as Hadoop (sequential access, 64kB requests)
  - Minimal CPU overhead
- Non-virtualized host domain for comparison purposes
Eucalyptus Storage Performance

• The default configuration performs poorly
  • Have we already solved this problem?
  • Did Eucalyptus just choose some poor defaults?

• Expanded the test scope
  • Hypervisors: KVM, Xen
  • I/O virtualization:
    • Full virtualization: SCSI device
    • Para-virtualization: Virtio (for KVM), XVD (for Xen)
  • Sparse file, full file, and full disk backing options

• More data beyond application bandwidth
  • Disk request size, queue depth, and utilization
  • Measured in host domain to quantify disk efficiency
Write Bandwidth / Local Storage

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
<th>Request Size (kB)</th>
<th>Queue Size (Elements)</th>
<th>% Util Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>111</td>
<td>512</td>
<td>140</td>
<td>100%</td>
</tr>
<tr>
<td>KVM (*)</td>
<td>SCSI / <strong>sparse file</strong></td>
<td>1.3</td>
<td>15</td>
<td>0.9</td>
<td>90%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / <strong>full file</strong></td>
<td>62.6</td>
<td>128</td>
<td>0.82</td>
<td>81%</td>
</tr>
</tbody>
</table>

(*) = Default configuration

- Causes of poor initial write performance
  - Sparse file backing / expansion overhead
  - Small (15kB) disk requests
- Pre-allocating backing file on disk increases bandwidth
- Tradeoff – Starting guests takes much longer
## Write Bandwidth / Local Storage

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
<th>Request Size (kB)</th>
<th>Queue Size (Elements)</th>
<th>% Util Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>111</td>
<td>512</td>
<td>140</td>
<td>100%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / full file</td>
<td>62.6</td>
<td>128</td>
<td>0.82</td>
<td>81%</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio / full file</td>
<td>87.0</td>
<td>490</td>
<td>42</td>
<td>100%</td>
</tr>
</tbody>
</table>

- **Para-virtualized drivers** increase disk efficiency in KVM and Xen (not shown)
- Multiple outstanding requests (> 3.0)
  - Synchronous writes are committed to guest page cache and immediately return
  - Requests queued in OS and committed to disk in a batch
- Tradeoff - Requires guest OS support

+ Large requests (> 350kB)
  + Page cache in guest domain and/or device driver aggregates multiple 64kB application requests

March 13th 2010
### Write Bandwidth / Local Storage

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
<th>Request Size (kB)</th>
<th>Queue Size (Elements)</th>
<th>% Util Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>111</td>
<td>512</td>
<td>140</td>
<td>100%</td>
</tr>
<tr>
<td>KVM (*)</td>
<td>SCSI / sparse file</td>
<td>1.3</td>
<td>15</td>
<td>0.9</td>
<td>90%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / full file</td>
<td>62.6</td>
<td>128</td>
<td>0.82</td>
<td>81%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / disk</td>
<td>71.5</td>
<td>128</td>
<td>0.57</td>
<td>64%</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio / full file</td>
<td>87.0</td>
<td>490</td>
<td>42</td>
<td>100%</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio / disk</td>
<td>110</td>
<td>512</td>
<td>60</td>
<td>100%</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI / full file</td>
<td>58.4</td>
<td>498</td>
<td>142</td>
<td>100%</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI / disk</td>
<td>65.8</td>
<td>126</td>
<td>0.87</td>
<td>86%</td>
</tr>
<tr>
<td>Xen</td>
<td>XVD / disk</td>
<td>102</td>
<td>350</td>
<td>3.0</td>
<td>100%</td>
</tr>
</tbody>
</table>

(*) = Default configuration
Write Bandwidth / Local Storage

- Full disk backing improves performance further over file backing
  - Para-virtualized drivers (in KVM) **comes within 1%** of non-virtualized disk bandwidth

- Tradeoff in flexibility – only one guest domain per disk partition
  - Acceptable for data-intensive computing applications
    - Storage performance is critical

- General-purpose cloud computing applications can continue to use file backing
  - Storage performance less critical
  - Sharing hardware between multiple guests is necessary for economic reasons
### Read Bandwidth / Local Storage

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
<th>Request Size (kB)</th>
<th>Queue Size (Elements)</th>
<th>% Util Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>108</td>
<td>256</td>
<td>0.94</td>
<td>96%</td>
</tr>
<tr>
<td>KVM (*)</td>
<td>SCSI / sparse file</td>
<td>71.9</td>
<td>225</td>
<td>1.1</td>
<td>96%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / full file</td>
<td>71.4</td>
<td>241</td>
<td>0.64</td>
<td>64%</td>
</tr>
<tr>
<td>KVM</td>
<td>SCSI / disk</td>
<td>70.5</td>
<td>256</td>
<td>0.7</td>
<td>68%</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio / full file</td>
<td>75.9</td>
<td>256</td>
<td>0.7</td>
<td>69%</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio / disk</td>
<td>76.2</td>
<td>256</td>
<td>0.5</td>
<td>57%</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI / full file</td>
<td>83.1</td>
<td>121</td>
<td>1.6</td>
<td>99%</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI / disk</td>
<td>42.8</td>
<td>7</td>
<td>22.4</td>
<td>99%</td>
</tr>
<tr>
<td>Xen</td>
<td>XVD / disk</td>
<td>94.8</td>
<td>64</td>
<td>2.2</td>
<td>99%</td>
</tr>
</tbody>
</table>

(*) = Default configuration
Read Bandwidth / Local Storage

- Unable to reach peak disk read bandwidth
  - Best observed configuration (Xen / XVD) has a 12% performance gap in this best-case test
  - Average configuration has a 30% performance gap

- Disk access patterns show the problem. Either:
  - Small request sizes (< 7kB) – Disk is used inefficiently
  - Small queue depths (< 0.7 requests) – Disk sits idle waiting for requests

- Challenge with synchronous I/O – application issues a 64kB read request and then waits for the data
  - Guest OS page cache may pre-fetch amount of additional data
Read Bandwidth

- Asynchronous I/O a solution for data-intensive computing?
  - Application can post many large read requests simultaneously

- Challenges in Hadoop / Linux
  - Asynchronous I/O in Linux only works in conjunction with O_DIRECT mode (bypasses page cache)
  - Neither feature is natively supported in Java
    - But we only have to implement it once!
Network Storage

• Local storage suitable for scratch purposes only
  • Example: storing temporary map/reduce keys
  • Deleted when guests are stopped

• Network storage necessary for persistent data in cloud environment

• Performance in host domain:

<table>
<thead>
<tr>
<th>DD Application</th>
<th>Bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>65.2</td>
</tr>
<tr>
<td>Read</td>
<td>55.8</td>
</tr>
</tbody>
</table>
Network Storage

• Network storage bandwidth limited by ATA over Ethernet protocol
  • Degrades raw disk bandwidth by 40%+ just reaching the host domain
  • Simple request/response design, just like native ATA

• Potential optimizations not used in default Eucalyptus
  • Jumbo Ethernet frames (increase payload size of each ATA request)
  • At server application
    • Aggregate adjacent I/O requests to improve disk efficiency
    • O_DIRECT decreases CPU overhead

• What happens to bandwidth in the virtualized domain?
Bandwidth / Network Storage

- Virtualization increases latency to reach host domain and network driver, and degrades **write** bandwidth further

### Write

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>65.2</td>
</tr>
<tr>
<td>KVM (*)</td>
<td>SCSI</td>
<td>19.9</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio</td>
<td>20.3</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI</td>
<td>26.5</td>
</tr>
</tbody>
</table>

### Read

<table>
<thead>
<tr>
<th>VMM</th>
<th>Driver</th>
<th>Bandwidth (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>55.8</td>
</tr>
<tr>
<td>KVM (*)</td>
<td>SCSI</td>
<td>49.5</td>
</tr>
<tr>
<td>KVM</td>
<td>Virtio</td>
<td>48.0</td>
</tr>
<tr>
<td>Xen</td>
<td>SCSI</td>
<td>51.4</td>
</tr>
</tbody>
</table>

(*) = Default configuration
Conclusions – Hadoop Summary

![Bar chart showing bandwidth (MB/s) for different configurations: Local Host, Local KVM (SCSI), Local KVM (Virtio), Local Xen (SCSI), Network KVM, and Network Xen. The chart compares write and read operations, with percentage decreases indicated for each configuration.

- Local Host: -40%
- Local KVM (SCSI): -20%
- Local KVM (Virtio): -80%
- Local Xen (SCSI): -50%
- Network KVM: -66%

March 13th 2010
Conclusions

• Cloud computing framework degrades data-intensive computing applications significantly

• Configuration changes improve out-of-box performance while still maintaining API compatibility
  • Full backing files instead of sparse
  • Para-virtualized block I/O instead of fully-virtualized

• Future work needed to close performance gap
  • Improve network disk protocol implementation
  • Explore impact of asynchronous I/O on virtualized guest performance
Questions?