ECE7995
(8) QoS for Cloud Computing
Cloud Computing

- Software/hardware as a Service.
  - Data, software application, and computer processing power as a cloud of on-line resources.
- Treat computing as utility service.
  - Elastic, flexible, inexpensive, and reliable.
Cloud Computing

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)
Data-intensive Applications using Consolidated Storage Systems

- Applications become more data intensive
  - Scientific applications may analyze large data sets.
  - Internet search and E-commerce rely on efficient data access.
  - Applications’ performance highly depends on I/O service quality.

- Advantages of consolidated storage system
  - High utilization due to resource sharing.
  - Cost-effectiveness of centralized management.
  - Lower operating cost.

- Each user essentially reserves a virtual storage device.
  - Contractual quality of services (QoS) requirements (SLA).
  - How to specify the I/O QoS requirements?
Amazon EC2 is a web-based service that provides resizable computing capacity in the cloud.

- Create an Amazon Machine Image (AMI), including your programs and data.
- Upload the AMI into Amazon data storage facility.
- Choose the computing capacity (instance type).

### An Example Issue: Amazon Elastic Compute Cloud (Amazon EC2)

**Equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.**

<table>
<thead>
<tr>
<th>Available Instance Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard Instances</strong></td>
</tr>
<tr>
<td><strong>Small Instance (default)</strong></td>
</tr>
<tr>
<td>1.7 GB memory</td>
</tr>
<tr>
<td>1 EC2 Compute Unit (1 virtual core with 32-bit platform)</td>
</tr>
<tr>
<td>I/O Performance: Moderate</td>
</tr>
<tr>
<td>Price: $0.10 per instance hour</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Large Instance</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5 GB memory</td>
</tr>
<tr>
<td>4 EC2 Compute Units (2 virtual cores with 64-bit platform)</td>
</tr>
<tr>
<td>I/O Performance: High</td>
</tr>
<tr>
<td>Price: $0.40 per instance hour</td>
</tr>
</tbody>
</table>
Service Level Agreement
(e.g., response time < 100ms, throughput > 100 trans./s)

100ms or 10ms latency?

100MB/s or 10MB/s throughout?
Issues with the Use of Fixed I/O Bounds

- **I/O intensity can change from time to time.**
  - Requests in the burst period share the same latency bound with those in the quiet period?
  - If the bound is determined according to requests in the quiet period, how much resources are demanded to meet it during the busy period?

- **Request size can be highly variable.**
  - One common latency bound for small and large requests?
  - If the throughput is in form of MB/s, any incentive to aggregate small requests into one large one?

- **Spatial locality of requests can vary substantially.**
  - One common throughput bound for random and sequential requests?
  - Shall the bound be determined according to random requests or sequential ones?
Implications of Fixed I/O Bounds

- They may not reflect applications’ real QoS needs.

- They may discourage programmers’ efforts on the optimization of I/O requests.

- They can pose highly variable resource demands on the storage system.
Our Solution: Use Reference Storage System as Performance Interface

- Assume that a user can receive satisfactory application performance with use of a dedicated storage system.
  - He wants to keep the performance after outsourcing I/O service to a shared storage system.

- The dedicated storage system is used as its performance interface.
  - The interface is called Reference Storage System (RSS)
  - By implementing the interface, the user will receive performance at least as good as that received on the RSS.

- The RSS interface is not subject to variation of I/O behaviors.
  - The interface is tangible to end users and is more relevant to application performance.
  - The interface can easily bound the resource demand on the shared storage system.
Interconnect controller

Application server 1

Application server 2

iSCSI

HDD disk array

SSD disk array

Request Streams

QoS-aware Scheduler

Performance Interface

RSS

Dedicated Local Storage

Reference Storage System
Interconnection Fabric

Application server 1

Application server 2

......

Application server n

Request Streams

Performance Interface

QoS-aware Scheduler

HDD disk array

SSD disk array

Hybrid disk array

iSCSI

FibreChannel

RSS

Interface

VSD

controller

controller

controller
YouChoose: Implementation and challenges

- Interpret RSS for the I/O scheduler to implement the interface
  - Predict what the latency of an arriving request is if it was received by RSS.
  - It’s a challenge with different access patterns and system configurations.

- Efficiently implement the RSS interface.
  - Meet simultaneously RSS requirements for different VSDs
  - Able to exploit request locality for system efficiency.

- Migrate virtual storage devices (VSDs) for high device utilization.
  - Different disk arrays exhibit various efficiency in hosting VSDs.
  - Automatically place and migrate VSDs to host arrays for high efficiency.
Prediction with the CART Tool

- The CART (Classification And Regression Trees) Tool
  - Known for its efficiency and accuracy.
- Model Training

Request Feature Vector (request size, location, sequentiality, R/W)
Prediction with the CART Tool (cont’d)

- Use the Model

Request Feature Vector (request size, location, sequentiality, R/W)

Real Workload

Training Data

- Predicted Response Time (PRT)

Trained Storage Model

- PRT_1
- PRT_2
- PRT_3
- \ldots
- PRT_n
YouChoose Request Scheduling

- We can predict a request’s service time on RSS (ref_time).

- N+1 clocks:
  - One wall clock (wall_clock)
  - N reference clocks (ref_clock).

- When the stream is considered for scheduling:
  - If its request is dispatched, then
    \[ \text{ref\_clock} += \text{ref\_time}_{\text{req}} \]
  - No pending requests, then
    \[ \text{ref\_clock} = \text{wall\_clock}. \]
Serving Requests in Batches for Efficiency

Requested data spread over multiple disks
Performance Evaluation

- Disk arrays simulated by DiskSim
  - Fast disks: QUANTUM TORNADO (10025RPMs, 1.245ms)
  - Slow disks: SEAGATE ST32171W (7200RPMs, 1.943ms)

- Synthetic traces
  - Request size: 4KB
  - Spatial locality $x \% \in [0\%-100\%]$: the probability of two consecutive requests for contiguous data.

- Real-world I/O traces
  - Financial: traces from OLTP applications at two large financial institutions.
  - WebSearch: traces from a popular search engine.
  - OpenMail: collected on a production e-mail system running the HP OpenMail
  - VideoStreaming: collected when playing a movie (sequential access)
Accuracy of the RSS Interface interpreted by CART

*WebSearch*

![Graph showing Accuracy of the RSS Interface interpreted by CART](image)
Estimation Accuracy for Time Windows

- More than 85% of relative errors are smaller than 15%
- For Individual Requests
- For 0.04s Time Window
- For 0.08s Time Window
Impact of Spatial locality (on Dedicated RSS)
Impact of Spatial locality
(on Shared Storage w/ YouChoose)
Impact of Spatial locality 
(on Shared Storage using the 100 IOPS Bound)
Performance Isolation (on Dedicated RSS)

- **Workload 1**: 100% Locality
- **Workload 2**: 75% Locality
- **0% Locality**
Performance Isolation
(on Shared Storage w/ YouChoose)

![Graph showing throughput (IOPS) vs. time (seconds) for two workloads with different locality percentages. Workload 1 has 100% Locality, Workload 2 has 75% Locality, and a third workload has 0% Locality.]
Performance Isolation (on Shared Storage using the 100 IOPS Bound)
Performance Isolation
(Real-world workloads on dedicated RSS)
Performance Isolation (Real-world workloads)
U-Shape: Providing Convenient Performance Interface to HPC Apps

- **What is U-Shape?**
  - A new scheduling framework that derives and implements instantaneous throughput requirement to meet app’s performance.
  - Apps’ requirements are specified as their runtimes relative to theirs runs on dedicated storage system.

- **Objectives of U-Shape**
  - Automatically convert the end-user QoS requirements into instantaneous derived-throughput bounds at run time.
  - Not only meet QoS requirements but also optimize data access efficiency on storage system consisting of multiple data servers.
  - Tightly integrate the new performance interface with existing I/O scheduling of the storage system.
A Case Study

Experimental setting
- 4 compute nodes, 9 data servers, 1 meta-data server
- PVFS2-2.8.2 with default striping configuration
- MPICH2-1.1.1 compiled with ROMIO
- CFQ disk scheduler

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Dedicated Execution</th>
<th>Allowed Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi-io-test</td>
<td>50s</td>
<td>2X</td>
</tr>
<tr>
<td>ior-mpi-io</td>
<td>54s</td>
<td>2X</td>
</tr>
</tbody>
</table>
Scheduling using Average Throughput as Performance Bounds

<table>
<thead>
<tr>
<th>Workload</th>
<th>Required</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi-io-test</td>
<td>110s</td>
<td>154s</td>
</tr>
<tr>
<td>ior-mpi-io</td>
<td>108s</td>
<td>122s</td>
</tr>
</tbody>
</table>
Scheduling with U-Shape

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Required</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi-io-test</td>
<td>110s</td>
<td>110s</td>
</tr>
<tr>
<td>ior-mpi-io</td>
<td>108s</td>
<td>105s</td>
</tr>
</tbody>
</table>
Implementation of U-Shape

- Predict instantaneous throughout according to:
  - The proportion of I/O time in the runtime,
  - Required relative runtime
  - Scaled down ML model

- Time-window based scheduling
  - U-Shape divides wall-clock time into a series of scheduling periods.
  - Each period consists of a number of scheduling windows, each dedicated to one program.
  - Window size is adjusted according to predicted instantaneous throughout.
Implementation of U-Shape (cont’d)

- Coordination of request scheduling across data servers.
  - Built on the IOrchestrator infrastructure (SC’10)
Conclusions

- **Higher performance through:**
  - Exploiting both temporal locality and spatial locality.
  - Coordination of different layers of the I/O stack.

- **Better performance provision enabled by:**
  - Convenient performance interface.
  - Efficient implementation of the interface.