Introduction

Workload Modeling and Generation is important because:
• Replay of original application in all storage system configurations is impractical
• Datacenter Workloads are not publicly available
• Storage System ~ 20-30% of TCO and power consumption of the total system

GOAL: Design a tool that recreates representative datacenter I/O workloads with high fidelity

APPLICABILITY: SSD Caching, Defragmentation Benefits, Storage Consolidation, ...

NOTE: Generation of the I/O access patterns NOT the application’s functionality

Two Step Approach

1-2: Traces to Models
• TRACES ETW*
  Information: Block offset, Type of I/O, File name, Number of Thread, Disk Number

MODELS
• Create models of one or multiple levels per app

SYNTHETIC WORKLOADS
• Generate a storage load that resembles the original app

Two Step Modeling-Generation Approach

*Event Tracing for Windows

Model

State Diagram-Based Probabilistic Model:
• State: Logical block range on disk
• Transition: Probability of switching between block ranges
• Stats: rd/wr, rmd/seq, block size, inter-arrival time

Figure 2: Simple State Diagram (1 level)

Extend the simple, one level model to a hierarchical representation.

Choose an optimal number of levels per application

Figure 3: Hierarchical State Diagram (2 levels)

Reduce Model Complexity: Spatial Locality within a state rather than across states (Hierarchical rather than Flat representation)

Previous Tools

IOMeter is the most well-known open-source workload generator

<table>
<thead>
<tr>
<th>Feature</th>
<th>Synthetic</th>
<th>IOMeter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-arrival Time (mean or distribution)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intensity Knob</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spatial Locality</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Trace Replay</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Different Levels of Granularity</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 1: IOMeter – DiskSpd Comparison

Implementation

1/4: Inter-Arrival Times

Inter-arrival Time: The time between two subsequent I/O requests.

Inter-arrival Times # Outstanding I/Os

Generating inter-arrival times both static and with time distributions

Figure 4: Normal and Exponential Inter-arrival Time Distributions

2/4: Thread Weights

One thread = One transition in the state diagram

Specific I/O characteristics per thread.

Thread Weight: The proportion of accesses for one transition. Thread weights are satisfied with less than 0.05% deviation

3/4: Intensity Knob

Evaluation of different storage system configurations (Disk vs. SSD)

Scale the inter-arrival times (more or less intense workload) without retuning the application

Replication of the exact same I/O request (block offset, type, block size)

Applicability: Error Detection in large-scale DBs.

Model & Tool Validation

• Collect Traces of Original Applications
• Create One/Multiple Level State Diagrams
• Compare I/O characteristics and Performance Metrics between Original and Synthetic Traces

Applicability: Error Detection in large-scale DBs.

Applicability: Model & Tool Validation

• Model and Generate representative DC storage I/O loads with high fidelity and density in time
• Use the tool to motivate two important challenges in DC storage system design: SSD caching and the benefits from Defragmentation without the requirement for access to app code or full application deployment

FUTURE WORK:
• Evaluate energy efficiency for SSD caching and defragmentation
• Expand a similar methodology to other parts of the system to create a Complete Workload Model with applications in virtualization, etc.

Figure 5: Validation of Throughput

Figure 7: Storage Speedup and Power Savings from Defragmentation

Figure 6: DiskSpd – IOMeter Comparison. Using IOMeter either has NO SPEEDUP (6-a) or INCONSISTENT SPEEDUP (6-b) with increasing number of SSDs

2. Defragmentation Benefits

Random > 80% - Sequential < 20% for most DC applications

Performing Defragmentation during low throughput requirement phases improves performance/efficiency

Conclusions and Future Work

Application’s datacenter I/O workloads with high fidelity

Workload Modeling and Generation is important because:
• Replay of original application in all storage system configurations is impractical
• Datacenter Workloads are not publicly available
• Storage System ~ 20-30% of TCO and power consumption of the total system

GOAL: Design a tool that recreates representative datacenter I/O workloads with high fidelity

APPLICABILITY: SSD Caching, Defragmentation Benefits, Storage Consolidation, ...

NOTE: Generation of the I/O access patterns NOT the application’s functionality

Two Step Approach

1-2: Traces to Models
• TRACES ETW*
  Information: Block offset, Type of I/O, File name, Number of Thread, Disk Number

MODELS
• Create models of one or multiple levels per app

SYNTHETIC WORKLOADS
• Generate a storage load that resembles the original app

Two Step Modeling-Generation Approach

*Event Tracing for Windows

Model

State Diagram-Based Probabilistic Model:
• State: Logical block range on disk
• Transition: Probability of switching between block ranges
• Stats: rd/wr, rmd/seq, block size, inter-arrival time

Figure 2: Simple State Diagram (1 level)

Extend the simple, one level model to a hierarchical representation.

Choose an optimal number of levels per application

Figure 3: Hierarchical State Diagram (2 levels)

Reduce Model Complexity: Spatial Locality within a state rather than across states (Hierarchical rather than Flat representation)

Previous Tools

IOMeter is the most well-known open-source workload generator

<table>
<thead>
<tr>
<th>Feature</th>
<th>Synthetic</th>
<th>IOMeter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-arrival Time (mean or distribution)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intensity Knob</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Spatial Locality</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Trace Replay</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Different Levels of Granularity</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 1: IOMeter – DiskSpd Comparison

Implementation

1/4: Inter-Arrival Times

Inter-arrival Time: The time between two subsequent I/O requests.

Inter-arrival Times # Outstanding I/Os

Generating inter-arrival times both static and with time distributions

Figure 4: Normal and Exponential Inter-arrival Time Distributions

2/4: Thread Weights

One thread = One transition in the state diagram

Specific I/O characteristics per thread.

Thread Weight: The proportion of accesses for one transition. Thread weights are satisfied with less than 0.05% deviation

3/4: Intensity Knob

Evaluation of different storage system configurations (Disk vs. SSD)

Scale the inter-arrival times (more or less intense workload) without retuning the application

Replication of the exact same I/O request (block offset, type, block size)

Applicability: Error Detection in large-scale DBs.

Model & Tool Validation

• Collect Traces of Original Applications
• Create One/Multiple Level State Diagrams
• Compare I/O characteristics and Performance Metrics between Original and Synthetic Traces

Applicability: Error Detection in large-scale DBs.

Applicability: Model & Tool Validation

• Model and Generate representative DC storage I/O loads with high fidelity and density in time
• Use the tool to motivate two important challenges in DC storage system design: SSD caching and the benefits from Defragmentation without the requirement for access to app code or full application deployment

FUTURE WORK:
• Evaluate energy efficiency for SSD caching and defragmentation
• Expand a similar methodology to other parts of the system to create a Complete Workload Model with applications in virtualization, etc.