Performance Analysis of Virtual Environments

Nikhil V Mishrikoti
(nikmys@cs.utah.edu)

Advisor: Dr. Rob Ricci
Co-Advisor: Anton Burtsev
Introduction

Motivation

Virtual Machines (VMs) becoming pervasive in data centers and academic institutions.

- Honouring SLAs. Promised vs Actual.
- Quantify impact of Virtualization.
- Making sense of performance data.
Introduction

Project Goals

• Empower user to investigate performance problems with as little inertia as possible.
Introduction

Project

• **Framework** to build tools for **performing fine grained analysis** of,
  
  • resource utilization
  
  • overheads and
  
  • **performance bottlenecks**, in virtual setups.
Agenda

• Introduction
• Xen Overview
• Data Collection
• Analysis Framework
• Analysis Algorithms
• Composibility
Introduction

Xen Overview

Data Collection

Analysis Framework

Analysis Algorithms

Composibility
Xen Overview

- Open source.
- Widely used. Ex Amazon EC2
Agenda

- Introduction
- Xen Overview
- Data Collection
- Analysis Framework
- Analysis Algorithms
- Composibility
Data Collection

Xentrace - Overview

Xentrace is a lightweight tracing utility that collects hypervisor and domain level events. Ships with Xen.

Diagram:

Xentrace logs

HDD

Dom 0

Dom 1

........

Dom n

Hypervisor Events

Domain Events

Domain Events

Xen

Hardware
Data Collection

Xentrace - Advantages

• Widely available since it ships with Xen.
• Easily extensible.
Data Collection

Xentrace - Data

• Not originally intended for performance.

• Xentrace collects enormous amounts of raw information.
  
  • E.x: data collected during a disk intensive load for 1 minute exceeds 700 MB

• Hence, chose as the source of performance data.
Data Collection

Xentrace - Details

- Event masks to selectively capture event data.
- Log data is in binary format.
- Additional events not provided by Xentrace, can be manually added and collected by inserting trace macros in Xen or domain source.
Data Collection

Xentrace - Event Format

uint32  uint64  uint32  uint32
CPU     tsc     ns     event_id

Physical CPU id

Timestamps - CPU clock cycles & nanoseconds

Trace Event type

Optional trace data
Data Collection

Xentrace - Limitations

• **xentrace_format**: binary to text.
  - E.x: 700 MB log has 20+ million lines of text

• Very difficult to manually peruse and,
  - identify performance problems.
  - gain high level overview of performance.
Agenda

• Introduction
• Xen Overview
• Data Collection
• Analysis Framework
• Analysis Algorithms
• Composibility
Analysis Framework

Architecture

Dom 0
- Xentrace

Dom 1
- Domain Events
- Hypervisor Events

Dom n
- Domain Events

Xen

Hardware

Xentrace logs

HDD

Reader

Analyses
Analysis Framework

Overview

• Implementation split in two components.
  • **Reader**: Parses binary log data **offline** and passes C-style structs to **Analyses** component.
  • **Analyses**: Algorithms consisting a group of handlers for different event types. Generate high level performance metrics like CPU utilization, disk i/o performance etc.
Agenda

• Introduction
• Xen Overview
• Data Collection
• Analysis Framework
  • Reader
  • Analyses
• Analysis Algorithms
• Composibility
Analysis Framework

Reader - Caveats

- Two caveats make parsing non-trivial.
  - Events collected in logs not completely ordered.
Analysis Framework

Reader - Unordered logs

CPU 0
Analysis Framework

Reader - Unordered logs

CPU 0 → ev1 ev2 ev3 ev4 ev5 ev6

Time
Analysis Framework

Reader - Unordered logs

CPU 0 → ev1 ev2 ev3 ev4 ev5 ev6

CPU 1 → ev1 ev2 ev3 ev4 ev5 ev6

CPU 2 → ev1 ev2 ev3 ev4 ev5 ev6

Time
Analysis Framework

Reader - Unordered logs

CPU 0 → ev1 ev2 ev3 ev4 ev5 ev6

CPU 1 → ev1 ev2 ev3 ev4 ev5 ev6

CPU 2 → ev1 ev2 ev3 ev4 ev5 ev6

Ordered

Time

Wednesday, July 31, 13
Analysis Framework

Reader - Unordered logs

CPU 0 → ev1 ev2 ev3 ev4 ev5 ev6
CPU 1 → ev1 ev2 ev3 ev4 ev5 ev6
CPU 2 → ev1 ev2 ev3 ev4 ev5 ev6

Ordered
Log

Time

Wednesday, July 31, 13
Analysis Framework

Reader - Unordered logs

Unordered logs

Sort?

Ordered logs
Analysis Framework

Reader - Unordered logs

Unordered large logs

Ordered large logs

External Sort?

d-way Merge Sort

Wednesday, July 31, 13
Analysis Framework

Reader - Unordered logs

Unordered large logs

Ordered large logs

d-way Merge

Min-Heap
Analysis Framework

Reader - Caveats

• Two caveats make parsing non-trivial.
  • Events collected in logs not completely ordered.
  • Loss of events during log collection.
Analysis Framework

Reader - Lost Records

- Events come in faster than they can be flushed to disk.
- Xentrace inserts a lost_record event in the logs.
- Interferes with analysis esp. time sensitive.
Analysis Framework

Reader - Lost Records

• Tried different approaches.
  • Increase buffer size.
  • Use Event masks when possible.
  • Fix Xentrace bug.

• Treat \texttt{lost\_records} as just another event.

• It’s handler notifies other event handlers in execution of its occurrence.

• They deal with it appropriately (discarding analysis, ignoring it completely etc.)
Agenda

- Introduction
- Xen Overview
- Data Collection
- Analysis Framework
  - Reader
    - Analyses
- Analysis Algorithms
- Composibility
Analyses

• Each tool’s analysis logic is made up of Event Handlers.

• Event Handlers registered with Reader.

• Handler needs 3 methods written by the user.
  • Initialize
  • Handle
  • Finalize
  • Ex: Count of Events. Initialize count to 0, increment count, print count at the end.
Analyses

Architecture

Xentrace logs

Reader

Parse Xentrace records

Call Event Handlers

Handlers

ev_handler_1(..)

ev_handler_2(..)

ev_handler_3(..)

Finalizers

ev_id_1

ev_id_2

ev_id_3

Free Handlers

Wednesday, July 31, 13
Agenda

- Introduction
- Xen Overview
- Data Collection
- Analysis Framework
  - Reader
  - Analyses
- Analysis Algorithms
- Composibility
Analysis Algorithms

- Reasoning about performance in virtual environments is *not* always straightforward.
- The user has to follow his instinct or data from another analysis.
Analysis Algorithms

Motivation

• Quantify phenomena observed on virtual setups.

• **Utilization Saturation Errors (USE)** methodology [1]
Analysis Algorithms

• CPU Utilization
• CPU Scheduling Latency
• Time in Hypervisor
• Disk I/O
  • Device Driver Queue Status
  • Device Driver Queue Request and Response Latency
Analysis Algorithms

CPU Utilization - Why?

- Fine grained CPU utilization information can,
  - Check if hypervisor adheres to Service Level Agreements (SLA) between hosting providers and clients.
  - Detecting unbalanced mapping between physical CPUs and VMs.
Analysis Algorithms

CPU Utilization

dom 0

vcpu0  vcpu1

v0  v1  v2

CPU 0

CPU 1

..............

CPU n

dom 1

dom U

vcpu0

Wednesday, July 31, 13
Analysis Algorithms

- CPU Utilization
- CPU Scheduling Latency
- Time in Hypervisor
- Disk I/O
  - Device Driver Queue Status
  - Device Driver Queue Request and Response Latency
Analysis Algorithms

CPU Scheduling Latency

`domain_wake(dom1, vcpu 0)`
`domain_wake(dom1, vcpu 1)`
`enter_sched(v0)`
`enter_sched(v1)`
Analysis Algorithms

CPU Scheduling Latency

• Measures time a VM had to wait to get scheduled since the context switch request was sent out.

• Delay in context switch not only affects CPU bound tasks but also I/O jobs.
  • Since domain needs CPU to process I/O requests/responses.
Wait times can increase for a number of reasons,

- VCPUs are over-scheduled.
- Physical CPU is always busy.
- Imbalance in VCPU => CPU affinity.
Analysis Algorithms

CPU Scheduling Latency

domId: 0 : CPU Wait Time: 32.677604 (ms)
domId: 1 : CPU Wait Time: 12.826167 (ms)

Total CPU Wait time for all domains: 45.503771 (ms)

0 - 700 (ns) : 0
700 - 1400 (ns) : 16178
1400 - 2100 (ns) : 12835
2100 - 2800 (ns) : 1384
2800 - 3500 (ns) : 28
3500 - 4200 (ns) : 1
4200 - 4900 (ns) : 0
4900 - 5600 (ns) : 0
5600 - 6300 (ns) : 0
6300 - 7000 (ns) : 0
> 7000 (ns) : 0
Analysis Algorithms

- CPU Utilization
- CPU Scheduling Latency
- Time in Hypervisor
- Disk I/O
  - Device Driver Queue Status
  - Device Driver Queue Request and Response Latency
Analysis Algorithms

Time in Hypervisor - Why?

- CPU utilization data can sometimes be unreliable to infer performance problems from.
- Possible case is when most execution takes place in hypervisor.
  - E.x: When significant amount of time is spent in the hypervisor, executing instructions or performing I/O on behalf of a domain, CPU util data will not show this behavior.
  - E.x: Honoring SLAs. Does SLA include Xen runtime?
Analysis Algorithms
Time in Hypervisor

Total of 0 lost_record events encountered

Total time spent in Domain 0 : CPU 1 = 307.576 (ms)
Total time spent in Domain IDLE : CPU 1 = 35333.512 (ms)
Total time spent in Domain 0 : CPU 0 = 731.296 (ms)
Total time spent in Domain IDLE : CPU 0 = 34436.609 (ms)
Total time spent in Domain 1 : CPU 2 = 19116.232 (ms)
Total time spent in Domain IDLE : CPU 2 = 16480.766 (ms)

Total time spent in Xen: 715.835 (ms)
Analysis Algorithms

• CPU Utilization
• CPU Scheduling Latency
• Time in Hypervisor
• Disk I/O
  • Device Driver Queue Status
  • Device Driver Queue Request and Response Latency
Analysis Algorithms

Disk I/O - Split Device Driver Model

Backend

Frontend

Xen

49
Analysis Algorithms

Disk I/O - Why?

• Performance impact of split device drivers on Disk I/O.
Analysis Algorithms

Disk I/O - Shared Ring Buffer [2]

1. DomU writes Req 1
2. DomU writes Req 2
3. Dom0 writes Resp 1
4. DomU reads Resp 1
5. Dom0 writes Resp 2
6. DomU reads Resp 2
Analysis Algorithms

Disk I/O - Simplified

Dom 0

Shared Ring buffer

Dom 1

Disk Request

Disk Response

Dom U

Event Mechanism

Xen

Hardware

HDD
Analysis Algorithms
Disk I/O - Device Driver Queues

Front Shared-Ring Request Queue

Back Shared-Ring Request Queue

Back Shared-Ring Response Queue

Front Shared-Ring Response Queue

Dom 0

Dom 1

BDD

FDD

Xen

Hardware

HDD
Analysis Algorithms
Disk I/O - Device Driver Queue States

• Blocked: Cannot process requests to/from queue.
  • Unable to add new requests to queue or
  • Queue is empty.

• Unblocked: Can enqueue new incoming requests.
Intuition was that a queue blocked for a long time would block the entire pipeline.
Analysis Algorithms

Disk I/O - Device Driver Queue States
Analysis Algorithms

Disk I/O - Device Driver Queue States

- Blocked
- Unblocked
- Unknown

Q_blocked

lost_records

Q_unblocked

Q_blocked

Q_unblocked

lost_records

Q_unblocked

lost_records

Q_unblocked
Analysis Algorithms

Disk I/O - Observations

• **Blocked**
  
  • Unbuffered: 99% - All queues
  
  • Buffered: **50%** - Frontend request queue, 99% rest.
    
    • Buffer cache enables faster request processing at frontend.

• Disk I/O so slow, virtualization overheads negligible.
Analysis Algorithms

- CPU Utilization
- CPU Scheduling Latency
- Time in Hypervisor
- Disk I/O
  - Device Driver Queue Status
  - Device Driver Queue Request and Response Latency
Analysis Algorithms

Queue Latency - Simplified

1. DomU writes Req & notifies Dom0
2. Dom0 reads Req
3. Dom0 writes Resp & notifies DomU
4. DomU reads Resp
Analysis Algorithms

Queue Latency - Simplified

1. DomU writes Req & notifies Dom0

2. Dom0 reads Req

3. SR Request Latency: $t_2 - t_1$

4. SR Response Latency: $t_4 - t_3$

5. Dom0 writes Resp & notifies DomU

6. DomU reads Resp
Analysis Algorithms

Disk I/O - Queue Latency

SR Request Latency

SR Response Latency

HDD

Disk Request

Disk Response

BDD

FDD

Dom 0

Dom 1

Xen

Hardware

Wednesday, July 31, 13
Analysis Algorithms

Queue Latency - Results

SR Response Latency $>>$ SR Request Latency

approx. 2 order of magnitudes greater for buffered i/o
## Queue Latency - Results

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Unblocked Time</th>
<th>Blocked Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front Request Queue Unblocked</td>
<td>1492.834 ms</td>
<td>1070.552 ms</td>
</tr>
<tr>
<td>Back Request Queue Unblocked</td>
<td>150.473 ms</td>
<td>3795.210 ms</td>
</tr>
<tr>
<td>Front Shared Ring Resp Queue Unblocked</td>
<td>108.448 ms</td>
<td>3837.193 ms</td>
</tr>
</tbody>
</table>

### QUEUE WAIT TIMES

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back Request Queue Wait Time</td>
<td>1.380 ms</td>
</tr>
<tr>
<td>Back Response Queue Wait Time</td>
<td>86.446 ms</td>
</tr>
</tbody>
</table>
Agenda

- Introduction
- Xen Overview
- Data Collection
- Analysis Framework
  - Reader
  - Analyses
- Analysis Algorithms
- Composibility
Composibility

Problem

Analysis: Average queue blocked times on domain 0
Composibility

Problem

Analysis: Average queue blocked times on domain 0

- Disk I/O analysis
- Averaging function
- Logic from CPU utilization
Composibility

Problem

Analysis: Average queue blocked times on domain 0

- Disk I/O analysis
- Averaging function
- Logic from CPU utilization

Lots of duplication of effort

New Tool
• Framework, so far, gives us stand alone tools for focussed analysis.

• Composibility gives agility to this framework.
Composibility

Overview

• Ability to reuse analysis algorithms logic for different event types.

• Easy to combine analysis tool outputs without having to rewrite large part of logic.

Compose new analysis using reusable parts.
Composibility

Overview

• Ability to reuse analysis algorithms logic for different event types.
  • Stages

• Easy to combine analysis tool outputs without having to rewrite large parts of logic.
  • Operators
Composibility

Pipeline

Analysis: Average queue blocked times on domain 0
Composibility

**Pipeline**

**Analysis**: Average queue blocked times on domain 0

**Stages**
- **dom 0 ?**
- **Disk I/O tool**
- **Average**

**Operators**
- **Pipeline**
- **output**
Composibility
Pipeline - Operators

• Pipe ( | ) : Connects a single stage to another.
• Split ( + ) : Connects a single stage to multiple stages. Executes all stages.
• Or ( or ) : Connects a single stage to multiple stages. Executes stages until valid return.
• Join : Connects multiple stages to a single stage. Either wait for a single connected stage to pass a valid event (JOIN_OR) or wait for all the connected stages to pass a successful event (JOIN_SPLIT).

Syntax ideas inspired from “A Universal Calculus for Streaming Processing Languages” [3]
Composibility
Pipeline - Operators Simplified

AND

OR

split

join_split

or

join_or
Composibility

Pipeline - Stages

- **Reusable** and **independent** analysis components.

- **Input:** One or more events.

- **Output:** *Same event, new event* with results from execution or **invalid event**.

- **If invalid** event returned, **break** from Pipeline.
Composibility

Pipeline

Analysis: Average queue blocked times on domain 0
Composibility

Pipeline

Analysis: Average queue blocked times on domain 0

event \[ \text{vm(dom0)} \] \rightarrow \text{Disk I/O tool} \rightarrow \text{Average} \rightarrow \text{output}
Composibility
Pipeline

Analysis: Average queue blocked times on domain 0

```
event → \text{vm(dom0)} → \text{wait\_time(Q\_B, Q\_U)} → \text{Average} → \text{output}
```
Composibility

Pipeline

Analysis: Average queue blocked times on domain 0

\[ \text{event_id}(Q_b) + \text{event} \rightarrow \text{vm(dom0)} \]

\[ \text{event_id}(Q_u) + \text{vm(dom0)} \rightarrow \text{wait_time()} \]

\[ \text{wait_time()} \rightarrow \text{Average} \]

\[ \text{Average} \rightarrow \text{output} \]

Wednesday, July 31, 13
Composibility

Pipeline

Analysis: Average queue blocked times on domain 0
Composibility
Pipeline - Syntax

vm(dom0) | event_id(Q_b) + event_id(Q_u) | wait_time() | average()
Composibility

Pipeline - Runtime

\texttt{vm(dom0) \mid event\_id(Q\_b) + event\_id(Q\_u) \mid wait\_time() \mid average()}

Parser [4]

\begin{verbatim}
s1 = create\_stage(vm, dom0);
s2 = create\_stage(event\_id, Q\_b);
s3 = create\_stage(event\_id, Q\_u);
s4 = create\_stage(wait\_time, NULL);
s5 = create\_stage(average, NULL);

split(s1, s2);
split(s1, s3);
join(s2, s4, JOIN\_SPLIT);
join(s3, s4, JOIN\_SPLIT);
pipe(s4, s5);

while(!feof(fp))
{
    parse\_next\_event(&ev);
    execute\_pipe(s1, ev);
}
\end{verbatim}
Composibility

Summary

• Reusable and independent analysis components - Stages

• Connect stages using Operators.

• Compose Pipeline using Stages and Operators
Demo

* If time permits ??
Conclusion

• **Goals met.**

• Easier to build tools for fine grained performance analysis of Xen - **Reader & Analyses**

• Build complex analysis tools in a short time - **Composibility**
Thank You

Q & A
References

• [1] USE method (http://dtrace.org/blogs/brendan/2012/02/29/the-use-method/)

• [2] Xen reference guide


88