vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities

June 18, 2009

Byung Chul Tak
Chunqiang Tang
Chun Zhang
Sriram Govindan
Bhuvan Urgaonkar
Rong N. Chang
Request-Processing Path Discovery

- Enterprise server architecture
  - Three-tiered architecture
    - Increasing complexity from heterogeneity
  - Virtualization

- Web Tier
  - VM1
  - VM2
  - VM3
  - LDAP Authentication

- Middle Tier
  - VM2
  - VM3
  - Dispatcher Thread
  - Stage 1
  - Stage 2
  - Stage 3

- Database Tier
  - HYPervisor

Multi-threaded Multi-staged Architecture
Motivation

🔹 *Request-Processing Path* information is critical to managing distributed applications
  - Debugging, analysis, auditing, billing …

🔹 Challenges in obtaining and exploiting the information
  - Develop application-specific middleware
  - Understand logs generated by the middleware
  - Pinpoint the root cause of the problem
Existing Solutions

❄ **Statistical inference**

*General, but NOT accurate, especially for individual Request-processing paths*

❄ **Instrumentation-based approach (E.g., Tivoli)**

*Requires app/MW/OS code changes*
vPath Technique

vPath discovers:

- **Precise** end-to-end request-processing path in a virtualized environment
- **Without instrumentation** on middleware or applications

Contributions

- New approach to the path discovery problem
  - Leverage common programming patterns in thread and communication
- Prototype implementation of the concepts
- Demonstration of accuracy and completeness
Key Concept of vPath

◆ Causal Relationships

Two types of causality

- Internal Causality
- External Causality
Source of Difficulty

- **Message flow**
  - Message is used up at arrival
  - Totally new message is assembled
    - Two messages share no common ID
  - Known options
    - Guess → statistical inference
    - Insert ID → instrumentation

- **Can we correlate incoming and outgoing messages?**
  - We consider the execution model
**Application Model**
- Thread pattern in Multi-threaded Model
  - Single thread is dedicated to the request until final response is sent out

**Communication Model**
- Synchronous communication
  - One thread sends a message and *blocks* until it receives the reply
Identifying Internal Causality

- Causality is carried on to the thread
- We identify thread from VMM

Identify the thread from the Virtual Machine Monitor
We use TCP socket information

(Source IP, port, Destination IP, port) is compared and connected

ex: [130.203.8.23:38294, 130.203.8.23:3314]

• We read socket information on Receive and Send events
Implementation

System Call Interception
- Intercept system calls by modifying Xen VMM
- For each system call, get thread identifier
  - EBP register value (stack address)

Diagram:
- CPU Hardware
- XEN VMM
- INT80h instruction
- INT80h Handler Pointer
- My Handler
- Interrupt descriptor table
- System Call Handler
- INT80h Handler
- user process
- kernel
- GUEST VM

Software trap
Socket info extraction

socket – (source IP, port, destination IP, port)
  • This uniquely identifies TCP connection
  • This enables us to correlate events across components

Custom hypercall
  • On every target system call, this hypercall is invoked
  • It delivers socket information from Guest Kernel to Xen VMM
Path Data Processing

◆ Log Format

From System Call Interception

<table>
<thead>
<tr>
<th>Event #</th>
<th>Domain #</th>
<th>Time Stamp</th>
<th>CR3</th>
<th>EBP</th>
<th>EAX</th>
<th>EBX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Hypercall

<table>
<thead>
<tr>
<th>Event #</th>
<th>OP Type (R/S)</th>
<th>Domain #</th>
<th>Socket Descriptor #</th>
<th>Local IP Addr &amp; Port</th>
<th>Remote IP Addr &amp; Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

0733  Dom1  002780  cr3:04254000  ebp:bfe37034  eax:3  ebx:12
0734  R   Dom1  sd:12   L:130.203.8.24:41845   R:130.203.8.25:8009
0735  Dom1  002781  cr3:04254000  ebp:bfe34b34  eax:146  ebx:11
0736  S   Dom1  sd:11   L:130.203.8.24:80   R:130.203.65.112:2395
0737  Dom2  002780  cr3:04254000  ebp:bff2203f  eax:3  ebx:12
0738  R   Dom2  sd:12   L:130.203.8.24:41811   R:130.203.8.25:8009
0739  Dom1  002781  cr3:04254000  ebp:bfe34b34  eax:146  ebx:11
0740  S   Dom1  sd:11   L:130.203.8.24:80   R:130.203.65.113:3411

Path Discovery Algorithm
vPath Prototype Components

Components

Online Monitoring Part
- System call interception at Xen VMM
  - Xen 3.1.0 for x86 32-bit Architecture
  - Guest Linux kernel 2.6.18
- Information collection for feeding to the analyzer

vPath Log Analyzer
- Algorithms for preprocessing
- Path construction logic
Evaluation Set-up

◆ Workloads
  - TPC-W – representing Java-based applications
  - RUBiS(PHP version)
  - vApp – custom C socket programming
  - MediaWiki

◆ System Set-up for TPC-W & RUBiS
  - Separate VMs for each application
Discovered Path for TPC-W

- Client Request
- Partial Reply
- Partial Reply
- Partial Reply
- Partial Reply
- Large Number of Requests and Replies between JBoss & MySQL
RUBiS

Discovered Path for RUBiS

- Client Request
- About 50 Consecutive recv()
- Sending Large Data Here
- Exactly 3 Round Trips

Client Request

Client Request

Reply

Reply
Overhead

vPath overhead on TPC-W response time

CDF of TPC-W Response Time

<table>
<thead>
<tr>
<th>Average</th>
<th>Response time (Degradation %)</th>
<th>Throughput (req/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla Xen</td>
<td>4.45</td>
<td>4.88</td>
</tr>
<tr>
<td>vPath</td>
<td>4.72 (6%)</td>
<td>4.59 (6%)</td>
</tr>
<tr>
<td>App Logging</td>
<td>10.31 (132%)</td>
<td>4.10 (16%)</td>
</tr>
</tbody>
</table>
Dissection of vPath Overhead

Worst case overhead measurement

Response Time (in Sec)

<table>
<thead>
<tr>
<th>Action</th>
<th>Vanilla Xen</th>
<th>System call Interception</th>
<th>Hypercall</th>
<th>Transferring Log to Dom0</th>
<th>Writing logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7% response time</td>
<td>0.7%</td>
<td>3.3%</td>
<td>19.3%</td>
<td>23.9%</td>
<td></td>
</tr>
</tbody>
</table>

Throughput (in req/sec)

<table>
<thead>
<tr>
<th>Action</th>
<th>Vanilla Xen</th>
<th>System call Interception</th>
<th>Hypercall</th>
<th>Transferring Log to Dom0</th>
<th>Writing logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>1.7%</td>
<td>4.5%</td>
<td>16.6%</td>
<td>19.1%</td>
<td></td>
</tr>
</tbody>
</table>
Limitations

- vPath works for Multi-threaded model
  - Unable to apply to event-driven or SEDA model
    - We argue that multi-threaded model is dominant

- Accessing socket information
  - Current implementation uses hypercall
    - Modification of the para-virtualized guest VM
    - Each system call incurs another mode-switch
Conclusion and Future work

 Proposal of vPath technique

- Accurate and non-intrusive technique of path discovery in a virtualized environment
- vPath exploits multi-threaded nature of applications and communication patterns
- Low run-time overhead

 Future Work

- Implementation of pure VMM-based approach
- More study on behaviors of various apps
- Collecting resource consumptions per path
Thank you