Benchmarking Dependability of Virtual and Cloud Environments

Samuel Kounev
Chair of Software Engineering
University of Würzburg

http://se.informatik.uni-wuerzburg.de/
Charles University, Prague, May 4, 2016
Explosion of IT Services & Users

Source: Gartner, IDC, Strategy Analytics, Machina Research, company filings, BI1 estimates
Explosion of Data

- 50x growth 2010-2020

Growing Number of Servers

- Google ~ 1 Mil. (2013)
- Microsoft ~ 1 Mil. (2013)
- Facebook ~ 180K (2012)
- OVH ~ 150K (2013)
- Akamai Tech. ~ 127K (2013)
- Rackspace ~ 94K (2013)
- 1&1 Internet ~ 70K (2010)
- eBay ~ 54K (2013)
- HP/EDS ~ 380K (2013)
- ...

Source: http://www.datacenterknowledge.com

Amazon’s Virginia region [Src: Wired.com]
Increasing Pressure to Raise Efficiency

- Proliferation of **shared IT infrastructures** → Cloud Computing
- Different forms of resource sharing (hardware and software)
  - Network, storage, and computing infrastructure
  - Software stacks

Datacenter Sharing

Virtualization
(e.g., as in IaaS)

Shared Middleware
(e.g., as in PaaS)

Multi-Tenancy
(e.g., as in SaaS)
Challenges

Expand / shrink resources on-the-fly

• When exactly should a reconfiguration be triggered?
• Which particular resources should be scaled?
• How quickly and at what granularity?
Challenges

Security Attack
Challenges

Hardware or Software Failure
Consequences

- **Challenges**
  - Increased system complexity and dynamics
  - Diverse vulnerabilities due to resource sharing

- **Service “dependability”** → major distinguishing factor between cloud platforms
  - Availability, reliability (+ security, performance, …)

- **Lack of reliable benchmarks to evaluate dependability**

“You can’t **control** what you can’t measure?” (DeMarco)

“If you cannot measure it, you cannot **improve** it” (Lord Kelvin)
Need for Benchmarks

“To measure is to know.” -- Clerk Maxwell, 1831-1879

“It is much easier to make measurements than to know exactly what you are measuring.” -- J.W.N.Sullivan (1928)

1. Reliable Metrics
   • What exactly should be measured and computed?

2. Representative Workloads
   • For which usage scenarios and under what conditions?

3. Sound Measurement Methodology
   • How should measurements be conducted?
The Focus of this Talk

- Metrics and benchmarks for quantitative evaluation of
  1. **Resource elasticity**
  2. Performance isolation
  3. Intrusion detection (and prevention)
- in shared execution environments
  - Virtualized infrastructures (e.g., as in IaaS)
  - Multi-tenant applications (e.g., as in SaaS)
Part I: Resource Elasticity

Main references


Further references


What People Say Elasticity Is…

- **ODCA, Compute Infrastructure-as-a-Service:**
  
  "[...] defines elasticity as the configurability and expandability of the solution[...] Centrally, it is the ability to scale up and scale down capacity based on subscriber workload."

- **NIST Definition of Cloud Computing**
  
  "Rapid elasticity: Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand."

- **IBM, Thoughts on Cloud, Edwin Schouten:**
  
  "Elasticity is basically a 'rename' of scalability [...]” and "removes any manual labor needed to increase or reduce capacity."

- **Rich Wolski, CTO, Eucalyptus:**
  
  "Elasticity measures the ability of the cloud to map a single user request to different resources."

- **Reuven Cohen:**
  
  Elasticity is "the quantifiable ability to manage, measure, predict and adapt responsiveness of an application based on real time demands placed on an infrastructure using a combination of local and remote computing resources."
What People Say Elasticity Is…

OCDA [1]
- up & down scaling
- with subscriber workload

IBM, Schouten [3]
- scalability
- increasing & reducing
- no manual labor

Cohen [5]
- quantifiable
- real-time demands
- local & remote

NIST [2]
- rapid elasticity
- unlimited
- provisioning & releasing
- sometimes automated
- with demand

Eukalyptus, Wolski [4]
- measurable
- mapping of
- requests to resources
Elasticity (in Cloud Computing)

Service Level Objective (SLO)
(e.g., resp. time ≤ 2 sec, 95%)

Resource Demand
Minimal amount of resources required to ensure SLOs

Workload intensity (e.g., # requests / sec)

Amount of resources (e.g., # VMs)

resource demand
underprovisioning
resource supply
overprovisioning
Def: The degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.

N. Herbst, S. Kounev and R. Reussner
Elasticity in Cloud Computing: What it is, and What it is Not.

Need for Benchmarks

“*To measure is to know.*” -- Clerk Maxwell, 1831-1879

“It is much easier to make *measurements* than to *know* exactly what you are measuring.” -- J.W.N. Sullivan (1928)

1. Reliable Metrics
   - What exactly should be measured and computed?

2. Representative Workloads
   - For which usage scenarios and under what conditions?

3. Sound Measurement Methodology
   - How should measurements be conducted?
Benchmarking Elasticity

1. Reliable Metrics
   • What exactly should be measured and computed?

2. Representative Workloads
   • For which usage scenarios and under what conditions?

3. Sound Measurement Methodology
   • How should measurements be conducted?
Metric 1: Accuracy

(1) accuracy_U: \( \frac{\sum U}{T} \)

(2) accuracy_O: \( \frac{\sum O}{T} \)
Same Metric Values - Different Behavior!

System A

System B
Metric 2: Timeshare

\( \text{timeshare}_U: \frac{\sum A}{T} \) \quad \text{(3)} \quad \text{timeshare}_O: \frac{\sum B}{T} \quad \text{(4)}
Same Metric Values - Different Behavior!

System A

System B

resources

resources

res. demand

res. supply

time

time
Metric 3: Jitter

\[
\text{jitter: } \frac{E_S - E_D}{T} \\
E_D: \# \text{ demand changes} \\
E_S: \# \text{ supply changes}
\]
Benchmarking Elasticity

1. Reliable Metrics
   • What exactly should be measured and computed?

2. Representative Workloads
   • For which usage scenarios and under what conditions?

3. Sound Measurement Methodology
   • How should measurements be conducted?
Example: Wikipedia Workload Trace

![Graph showing DLIM_wikipedia Arrival Rates](image-url)

Same Workload on Two Platforms

Resource demand
Resource supply
Underprovisioning

Same user workload on system B

platform A

platform B
Same Workload on Two Platforms

- **Resource demand**
- **Resource supply**
- **Underprovisioning**

Load intensity adjusted to induce the same demand curve as for platform A
Same Demand Variations on Two Platforms

- Red: Resource demand
- Blue: Resource supply
- Red with diagonal lines: Underprovisioning

System B at a user workload adjusted to induce the same demand curve.
Same Demand Variations on Two Platforms

- Resource demand
- Resource supply
- Underprovisioning

System B at a user workload adjusted to induce the same demand curve

Platform A

Platform B

Resource units [# VMs] vs. time
Benchmarking Elasticity

1. Reliable Metrics
   - What exactly should be measured and computed?

2. Representative Workloads
   - For which usage scenarios and under what conditions?

3. Sound Measurement Methodology
   - How should measurements be conducted?
Elasticity Benchmarking Approach

System Analysis
- Analyze efficiency & scaling behavior of underlying resources

Benchmark Calibration
- Adjust load profile

Measurement
- Expose SUT to varying load & monitor resource supply & demand

Metric Evaluation
- Compute elasticity metrics (accuracy & timing)

Case Study: Amazon Web Services vs. CloudStack

<table>
<thead>
<tr>
<th>Configuration</th>
<th>$\text{accuracy}_O$ [res. units]</th>
<th>$\text{accuracy}_U$ [res. units]</th>
<th>$\text{timeshare}_O$ [%]</th>
<th>$\text{timeshare}_U$ [%]</th>
<th>jitter [adap/min.]</th>
<th>elastic speedup</th>
<th>violations [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS – 1 Core</td>
<td>2.423</td>
<td>0.067</td>
<td>66.1</td>
<td>4.8</td>
<td>-0.067</td>
<td>1.046</td>
<td>7.6</td>
</tr>
<tr>
<td>CS – 2 Core adjusted</td>
<td>2.508</td>
<td>0.061</td>
<td>67.1</td>
<td>4.5</td>
<td>-0.044</td>
<td>1.025</td>
<td>8.2</td>
</tr>
<tr>
<td>AWS - m1.small</td>
<td>1.340</td>
<td>0.019</td>
<td>61.6</td>
<td>1.4</td>
<td>0.000</td>
<td>1.502</td>
<td>2.5</td>
</tr>
</tbody>
</table>
Part II: Performance Isolation

Main references


Further references


Example Scenario: Multi-Tenant Environments

Tenants working within their assigned quota (e.g., # users) should not suffer from tenants exceeding their quotas.
Definition of Performance Isolation

- Tenants working within their assigned quota (e.g., # users) should not suffer from tenants exceeding their quotas.
Definition of Performance Isolation

D is a set of **disruptive tenants** exceeding their quotas.

A is a set of **abiding tenants** not exceeding their quotas.
Performance Isolation Metrics

D is a set of **disruptive tenants** exceeding their quotas.

A is a set of **abiding tenants** not exceeding their quotas.

Approach: Quantify impact of increasing workload of the disruptive tenants on the performance of the abiding ones.
Metrics Based on QoS Impact

Reference Workload $W_{\text{ref}}$

Disruptive Workload $W_{\text{disr}}$

Load

Avg. response time for abiding tenants $A$

$\Delta w = \frac{\sum_{t \in W_{\text{disr}}} w_t - \sum_{t \in W_{\text{ref}}} w_t}{\sum_{t \in W_{\text{ref}}} w_t}$

$\Delta z_A = \frac{\sum_{t \in A} [z_t(W_{\text{disr}}) - z_t(W_{\text{ref}})]}{\sum_{t \in A} z_t(W_{\text{ref}})}$

Tenants

Workload
Example Metric

\[ I_{QoS} = \frac{\Delta z_A}{\Delta w} \]

Difference in response time

Difference in workload

Perfectly Isolated = 0
Non-Isolated = ?

Answers: How strong is a tenant’s influence on the others?
Metrics Based on Workload Ratio

- Workload vs. Time
- Response time vs. Time
For a given intensity of the disruptive workload, we plot the maximum possible intensity of the abiding workload, under which the QoS of the abiding tenants is maintained.
We can maintain the QoS for the abiding tenant without decreasing his workload.
Metrics Based on Workload Ratio

- Abiding workload
- Disruptive workload

- Isolated
- Observed system
- Non-isolated

$W_{a_{\text{ref}}}$
$W_{a_{\text{base}}}$
$W_{d_{\text{ref}}}$
$W_{d_{\text{base}}}$
$W_{d_{\text{end}}}$
Example Metric: $I_{\text{end}}$

Perfectly Isolated = ?
Non-Isolated = 0

Answers: How isolated is the system compared to a non-isolated system?
**Example Metric: $I_{\text{base}}$**

Perfectly Isolation = 1  
Non-Isolated = 0  
Describes the decrease of abiding workload at the point at which a non-isolated systems abiding load is 0.
Metrics Based on Workload Ratio Integrals

\[ W_{a_{\text{ref}}} \]

\[ W_{a_{\text{base}}} \]

\[ A_{\text{measured}} \]

\[ W_{d_{\text{ref}}} \]

\[ W_{d_{\text{base}}} \]

\[ W_{d_{\text{end}}} \]

Disruptive Workload

Observed System

Non-Isolated

Isolated

Abiding Workload
Metrics Based on Workload Ratio Integrals

\[ W_{a_{\text{ref}}} \]

\[ W_{a_{\text{base}}} \]

\[ W_{d_{\text{ref}}} \]

\[ W_{d_{\text{base}}} \]

\[ W_{d_{\text{end}}} \]

Isolated

Observed System

Non-Isolated

\[ A_{\text{nonIsolated}} \]
Metrics Based on Workload Ratio Integrals

- **Abiding workload**
- **Disruptive workload**

- $W_{a_{\text{ref}}}$
- $W_{a_{\text{base}}}$
- $W_{d_{\text{ref}}}$
- $W_{d_{\text{base}}}$
- $W_{d_{\text{end}}}$
- $p_{\text{end}}$

- Isolated
- Observed System
- Non-Isolated

$A_{\text{Isolated}}$
Metrics Based on Workload Ratio Integrals

\[ I = \frac{A_{\text{measured}} - A_{\text{nonIsolated}}}{A_{\text{isolated}} - A_{\text{nonIsolated}}} \]
Example Metrics: \( I_{\text{intBase}} \) and \( I_{\text{intFree}} \)

\[
I_{\text{intBase}} = \frac{\left( \frac{W_{d_{\text{base}}}}{W_{d_{\text{ref}}}} \int_{W_{d_{\text{ref}}}} f_{m}(W_{d}) dW_{d} \right) - W_{a_{\text{ref}}}^{2}/2}{W_{a_{\text{ref}}}/2}
\]

Areas within \( W_{d_{\text{ref}}} \) and \( W_{d_{\text{base}}} \)

\[
I_{\text{intFree}} = \frac{\left( \frac{p_{\text{end}}}{W_{d_{\text{ref}}}} \int_{W_{d_{\text{ref}}}} f_{m}(W_{d}) dW_{d} \right) - W_{a_{\text{ref}}}^{2}/2}{W_{a_{\text{ref}}} \cdot (p_{\text{end}} - W_{d_{\text{ref}}}) - W_{a_{\text{ref}}}^{2}/2}
\]

Areas within \( W_{d_{\text{ref}}} \) and predefined bound.

**Perfectly Isolated** = 1

**Non-Isolated** = 0

Answers: How much potential has the isolation method to improve?
Case Study

Add Delay

Round Robin

Blacklist

Separate Thread Pools

Part III: Intrusion Detection

Collaboration with
Marco Vieira and Nuno Antunes, University of Coimbra, Portugal
Bryan D. Payne, Department of Security Research, Nebula Inc.
Alberto Avritzer, Siemens Corporate Research, USA

Main references


Further references


IDS Evaluation

- Evaluation of intrusion detection systems (IDSes)
  - Enables the comparison of IDSes
  - Enables the improvement of the configuration of a deployed IDS

- IDSes for virtualized environments → many designs possible
  - Network intrusion detection by monitoring the virtual network bridge
  - Host intrusion detection through Virtual Machine Introspection (VMI)
Focus of our Work

IDS evaluation in virtualized environments

- Workloads
  - Injection of attacks targeting VMMs
    - Injection of representative hypercall attacks
- Metrics and measurement methodologies
  - New security-related metrics
    - Attack detection accuracy metrics that take elasticity into account

Metrics and measurement methodologies

- New security-related metrics
  - Attack detection accuracy metrics that take elasticity into account
Malicious Workloads: Generating Attacks

- **Focus:** VMMs as attack surfaces
  - Attack scenario: “malicious guest VM attacks the underlying VMM”
    - Attack vectors
      - Hypercalls
      - Routines / software traps invoked by kernels of paravirtualized, or HV with paravirtualized device(s), guest VMs for performing system management operations (e.g., sharing memory pages)

- **Hypercalls**
  - Vulnerabilities in VMMs’ hypercall handling routines are critical!
Malicious Workloads: Generating Attacks

- Defining representative/realistic attack scenarios
  - Attack models
    - Identify characteristics of hypercall attacks (e.g., specific hypercall parameter values, hypercall order, ….)
  - No attack scripts/proof-of-concept code available …
    - … however, patches are available!

- Approach:
  1. Select a set of hypercall vulnerabilities
  2. Reverse-engineer the patches of the selected vulnerabilities
     - 2.1 Develop proof-of-concept code
  3. Characterize hypercall attacks
Malicious Workloads: Generating Attacks

- **Artificial injection** of hypercall attacks based on **representative attack models**
  - Reason: Lack of publicly available attack scripts

- Attack models
  1. Analysis of relevant CVE reports
  2. Identification of patterns of VM activities
  3. Categorization of VM activity patterns into attack models

- Attack patterns
  1. Invoking hypercalls from irregular call sites
  2. Hypercalls with anomalous parameter values a) outside the valid value domains, or b) crafted for exploiting specific vulnerabilities (not necessarily outside the valid value domains)
  3. A series of hypercalls in irregular order, including repetitive execution of a single or multiple hypercalls

More later …
HInjector: Framework for Injecting Hypercall Attacks

Diagram:
- User -> Configuration
- Kernel
- Hardware
- IDS (in SVM)
- vCPU
- Memory
- Injector
- LKM
- Filter
- Hypercall handler

Arrows indicate flow or interaction between components.
Field Study on Hypercall Vulnerabilities

- **Goals**
  - Characterization and classification of hypercall vulnerabilities
  - Identification of causes of hypercall vulnerabilities
  - Provide technical information on hypercall vulnerabilities

- **Benefits**
  - Can we prevent future vulnerabilities?
    - Hypercall programming practices
    - Vulnerability discovery techniques
  - Can we detect and prevent the exploitation of existing vulnerabilities?
    - Hypercall attack detection and prevention mechanisms
## Field Study on Hypercall Vulnerabilities

<table>
<thead>
<tr>
<th>CVE</th>
<th>Hypercall</th>
<th>Vulnerable Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2012-3497 / CVE-2012-6036</td>
<td>tmem_op</td>
<td>&gt;= Xen 4.0.x</td>
</tr>
<tr>
<td>CVE-2012-5513</td>
<td>memory_op</td>
<td>&lt; Xen 4.1.4</td>
</tr>
<tr>
<td>CVE-2008-3687</td>
<td>flask_op</td>
<td>&lt; Xen 3.3</td>
</tr>
<tr>
<td>CVE-2013-0154</td>
<td>mmu_update</td>
<td>Xen 4.2.x</td>
</tr>
<tr>
<td>CVE-2013-1964</td>
<td>grant_table_op</td>
<td>Xen 4.1.x – 4.1.5</td>
</tr>
<tr>
<td>CVE-2012-4539</td>
<td>grant_table_op</td>
<td>Xen 4.1.x – 4.1.4</td>
</tr>
<tr>
<td>CVE-2012-5525</td>
<td>mmuext_op</td>
<td>Xen 4.2.x</td>
</tr>
<tr>
<td>CVE-2012-5515</td>
<td>memory_op</td>
<td>Xen 3.4.x – 4.1.4</td>
</tr>
<tr>
<td>CVE-2012-3494</td>
<td>set_debugreg</td>
<td>&lt; Xen 4.1.4 (4.1 ser.), Xen 4.2.0 (4.2 ser.)</td>
</tr>
<tr>
<td>CVE-2012-3496</td>
<td>memory_op</td>
<td>Xen 3.9.x – 4.1.4</td>
</tr>
<tr>
<td>CVE-2012-5514</td>
<td>memory_op</td>
<td>Xen 3.4.x – 4.1.4</td>
</tr>
<tr>
<td>CVE-2012-3495</td>
<td>physdev_op</td>
<td>Xen 4.1.x</td>
</tr>
<tr>
<td>CVE-2013-0154</td>
<td>mmuext_op</td>
<td>Xen 4.2.x</td>
</tr>
<tr>
<td>CVE-2012-5513</td>
<td>memory_op</td>
<td>Xen 4.1.x</td>
</tr>
<tr>
<td>CVE-2013-4553</td>
<td>domctl</td>
<td>&gt; Xen 3.4.x</td>
</tr>
<tr>
<td>CVE-2013-0151</td>
<td>hvm_op</td>
<td>Xen 4.2.x</td>
</tr>
<tr>
<td>CVE-2013-4494</td>
<td>grant_table_op</td>
<td>All versions of Xen up to the current date</td>
</tr>
<tr>
<td>CVE-2012-5510</td>
<td>grant_table_op</td>
<td>&lt; Xen 4.1.4 (4.1 ser.), Xen 4.2.0 (4.2 ser.)</td>
</tr>
</tbody>
</table>
Observations

- Errors causing hypercall vulnerabilities
  - Implementation errors (missing value validation, incorrect value validation, and incorrect implementation of inverse procedures)
  - Hypervisor design errors

- Most implementation errors are missing value validation errors
  - Internal variables (e.g., return codes)!
  - Eliminating missing value validation errors by adding program code verifying variable values →
    - Reduces hypercall execution speed → increased frequency of continuations → performance overhead →
    - Programming practices for boosting hypercall execution speed → vulnerabilities (e.g., CVE-2012-5535)
Summary

- Pressure to raise efficiency by sharing IT resources
- Resource sharing poses challenges
- Service “dependability” → major distinguishing factor
- Need for reliable benchmarks:
  - metrics, workloads and measurement methodologies
- **Multiple metrics** needed to understand system behavior
- **Choice of workloads** is critical for fair comparisons!
"It is easy to lie with statistics. It is hard to tell the truth without statistics."

-- Andrejs Dunkels

“I can prove it or disprove it! What do you want me to do?”
Thank You!

skounev@acm.org
http://se.informatik.uni-wuerzburg.de