Keeping an Eye on Virtualization: VM Monitoring Techniques & Applications

Validation

Why Is this Important ?

- Providing *a higher level of reliability and availability* is one of *the biggest challenges* of Cloud computing
  - Amazon's S3 outage (7/20/2008) due to *a single bit error in the gossip message*
    - *o*ffers customer 10% credit if S3 availability drops below 99.9% in a month
  - Google AppEngine's partial outage (6/17/2008) due to *a programming error*
  - Microsoft Azure's outage (3/17/2009) for 22 hours due to *the malfunction in the hypervisor*

Why Is this Important (cont)?

- **Virtualization** plays a role of enabling technology for the Cloud core architecture, e.g.,
  - Amazon's EC2 uses Xen hypervisor,
  - Microsoft's Azure uses Windows Azure Hypervisor
  - IBM cloud employs KVM hypervisor
- **Accidental failures and malicious attacks** can result in significant down time (lower availability) of systems and applications
- Important to characterize the **resiliency** of different cloud infrastructures

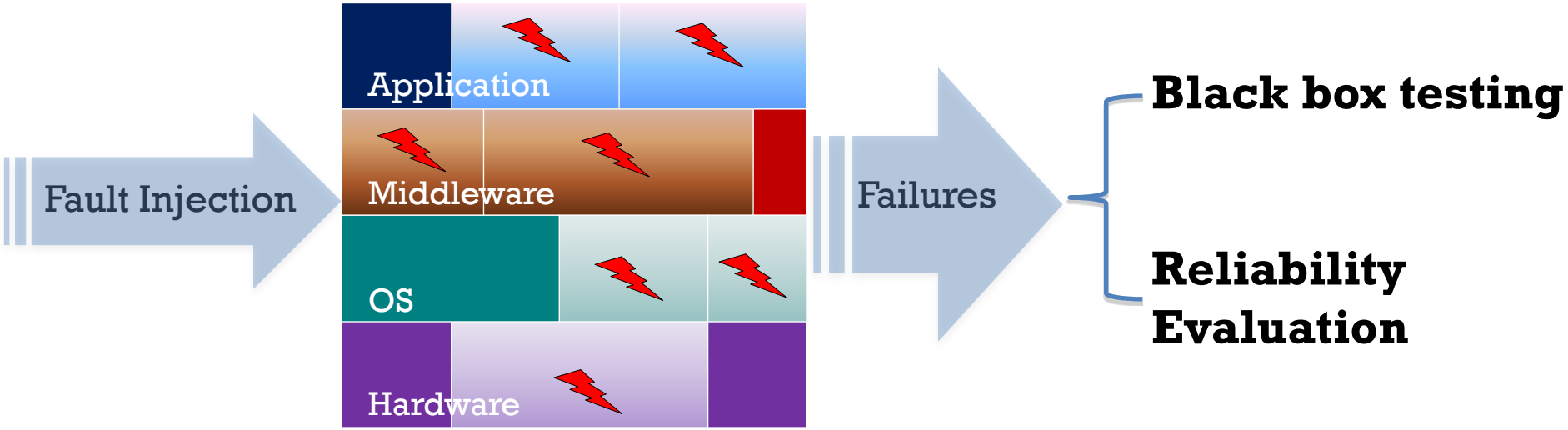
Outline

- Validation of virtualization environment in cloud infrastructure using fault injection
- Failure Diagnosis using message flow reconstruction and targeted fault injection

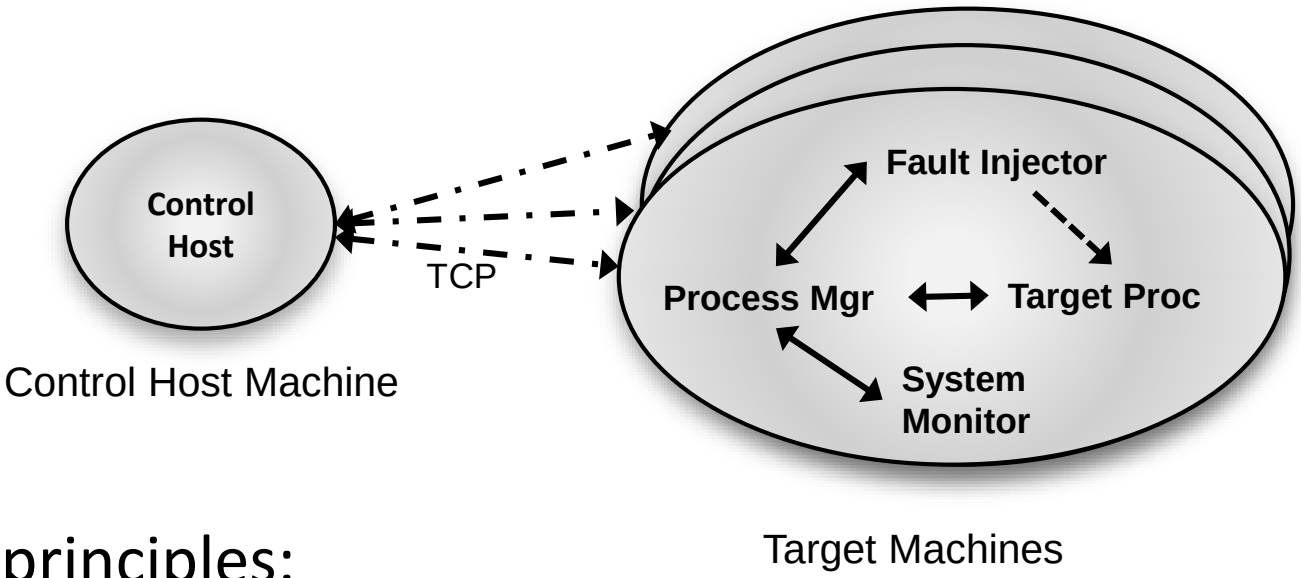
CloudVal: A Framework for Validation of Virtualization Environment in Cloud Infrastructure

Approach

Use a software implemented fault injection (SWIFI) framework (**CloudVal)** to automate the process of conducting **fault injection-based experiments** for **black box testing** and **reliability/security evaluation** of virtualized environments.



Fault Injection Framework

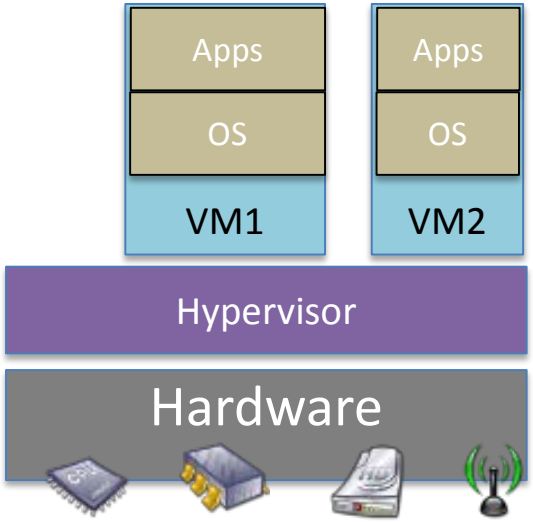


- Design principles:
  - Automatable, repeatable experiment
  - Realistic, representative, extendable fault models
  - Architecture independent, applicable for actual systems
- Extend NFTAPE fault injection framework

Stott, D.T., et al. , "NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors," *Computer Performance and Dependability Symposium, 2000. IPDS 2000*

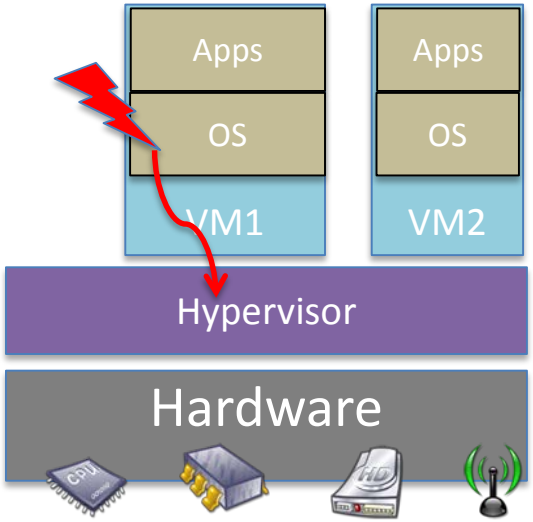
Fault Models

- Guest system misbehavior
- Soft errors
- Performance faults
- Maintenance faults (e.g., turn off a certain part of the hardware to reduce the power consumption)



FM 1: Guest system misbehavior

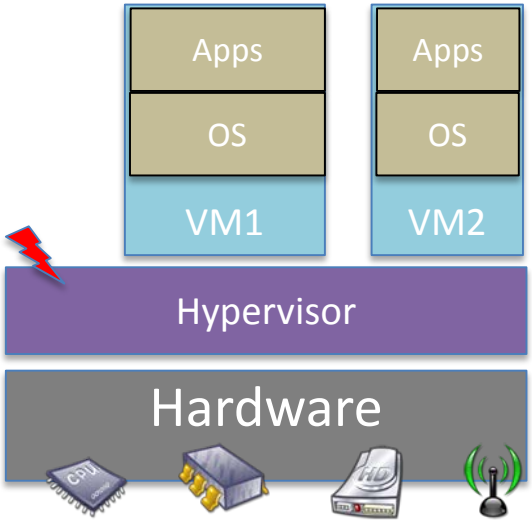
- Rationale
  - Test failure isolation between the guest system and the host system



- Simulation of the incident
  - Randomly corrupt memory of the kernel of the guest OS to generate failures in a guest VM

FM 2: Soft errors

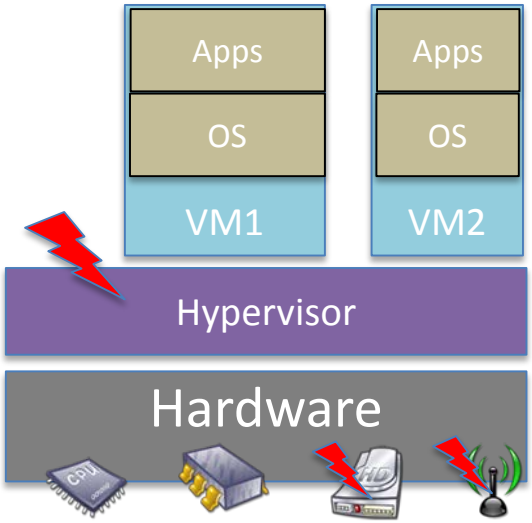
- Rationale
  - Test resiliency to single and multi-bit memory errors
  - Today's 1GB ECC protected memory has the same uncorrectable error rate as yesterday's 32MB parity protected memory



- Simulation of the incident:
  - Inject multi-bit errors into virtualization layers (user and kernel) and host OS (for hypervisor type 2)

FM 3: Performance faults

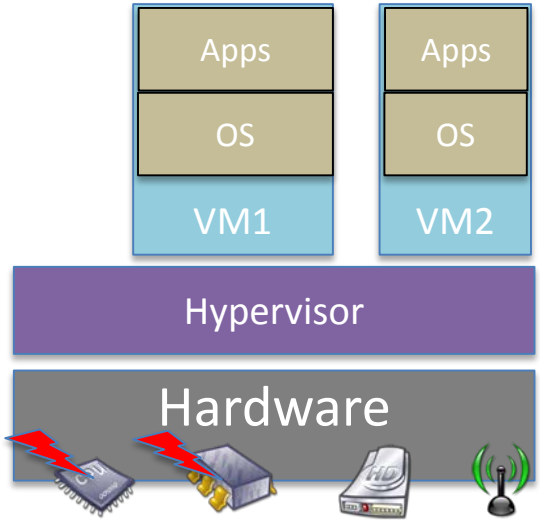
- Rationale
  - Virtualization facilitates sharing resources
  - Sharing can cause race condition
  - Simulate I/O delay, cpu exhaustion



- Simulation of the incident
  - Insert delay time in application threads

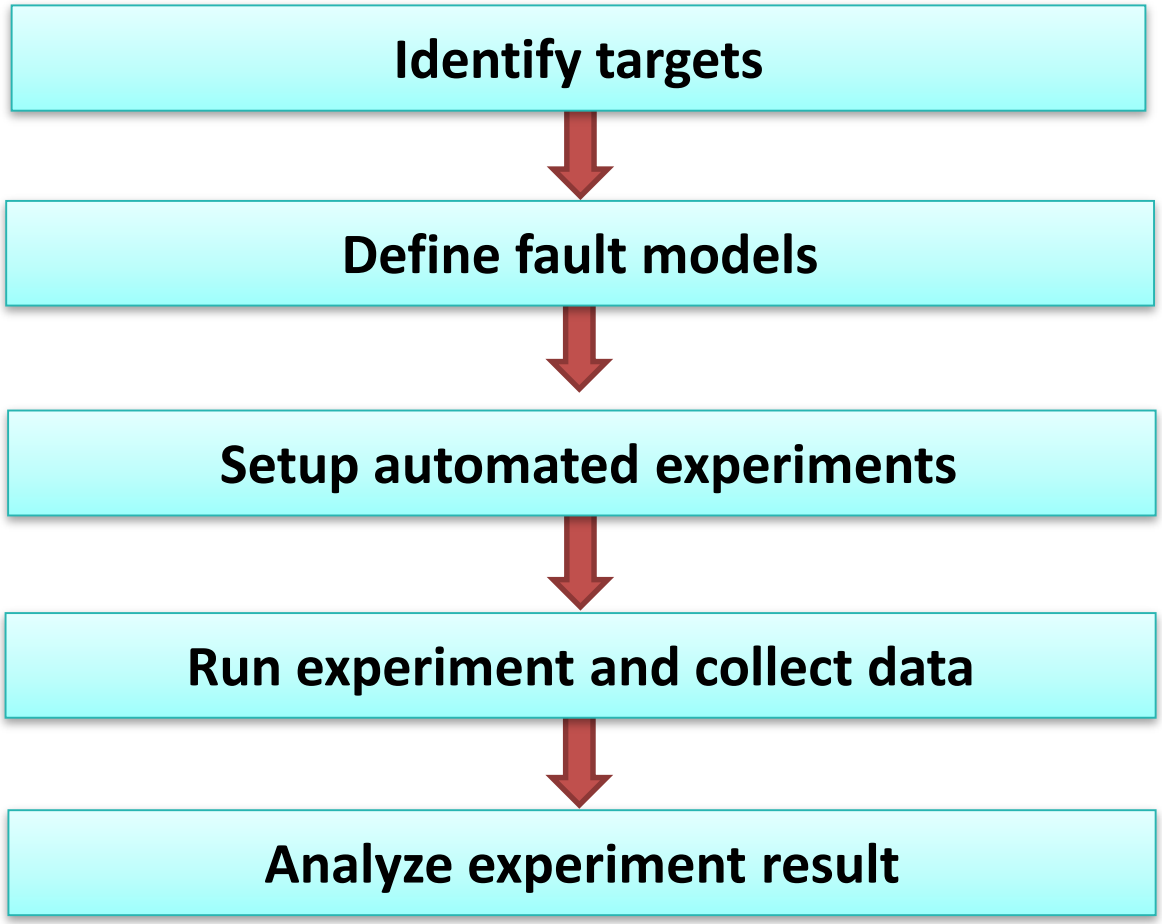
FM 4: Maintenance faults

- Rationale
  - Online replacement of hardware components due to hard errors
  - Turn off a certain part of the hardware to reduce the power consumption

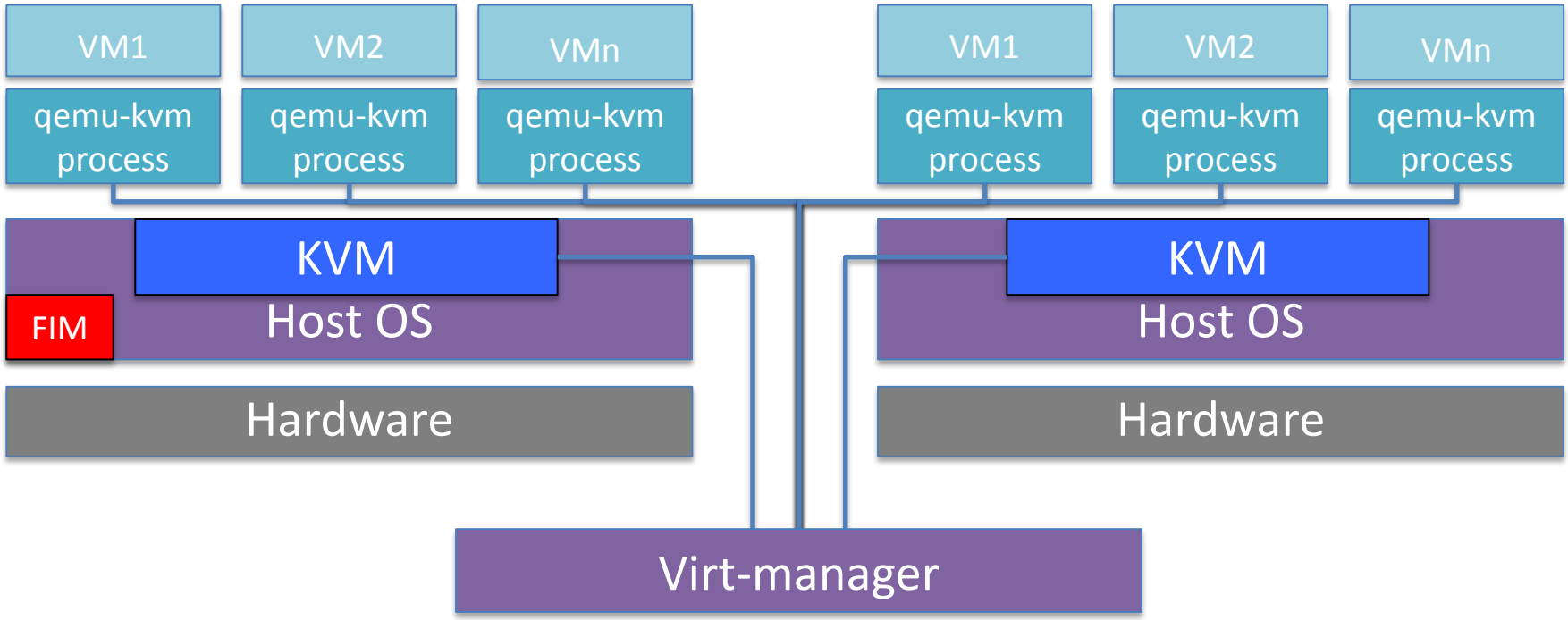


- Simulation of the incident
  - CPU hot-plug/unplug
  - Memory hot-add, hot-swap

Steps to conduct experiments

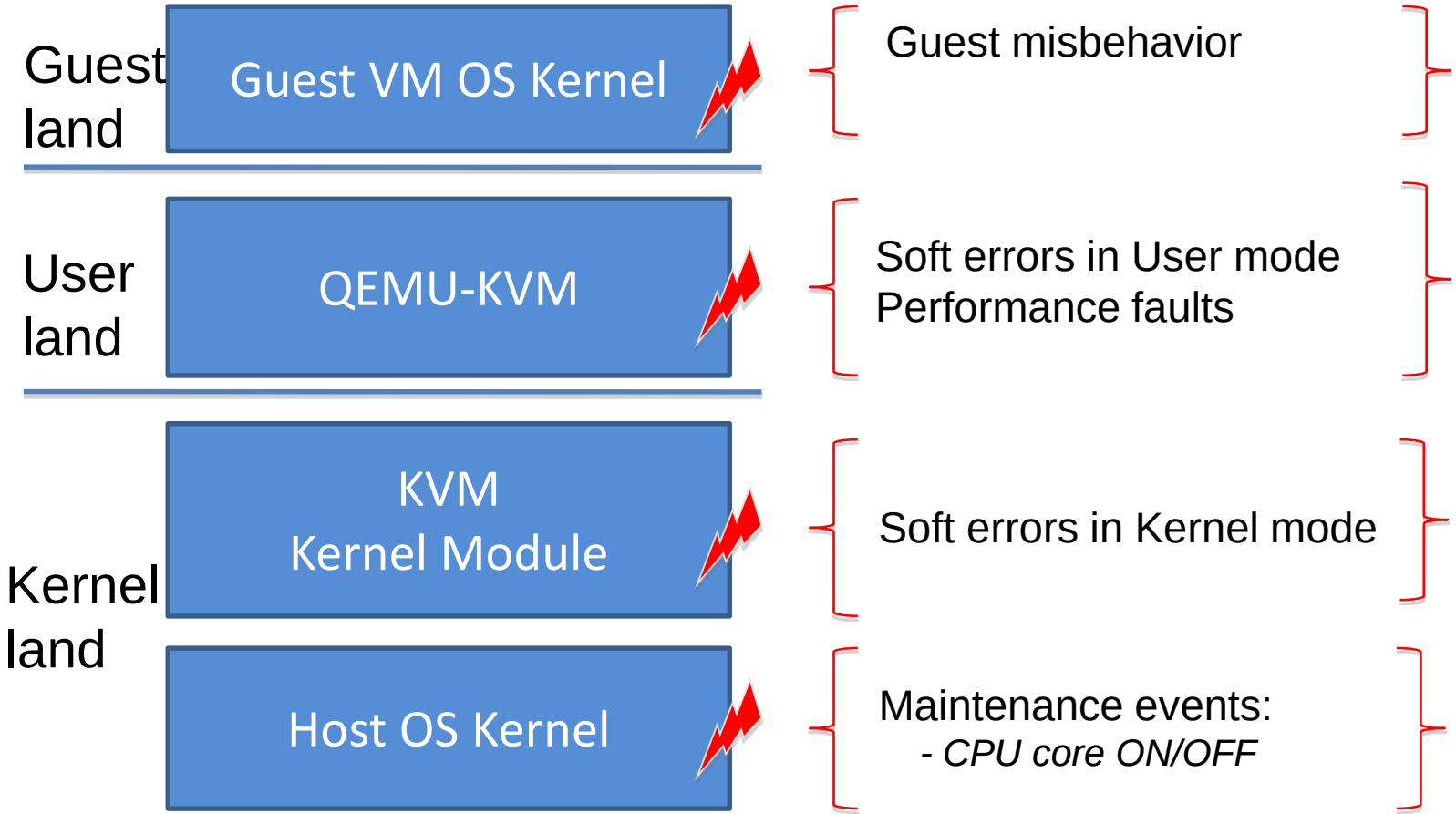


Identify targets

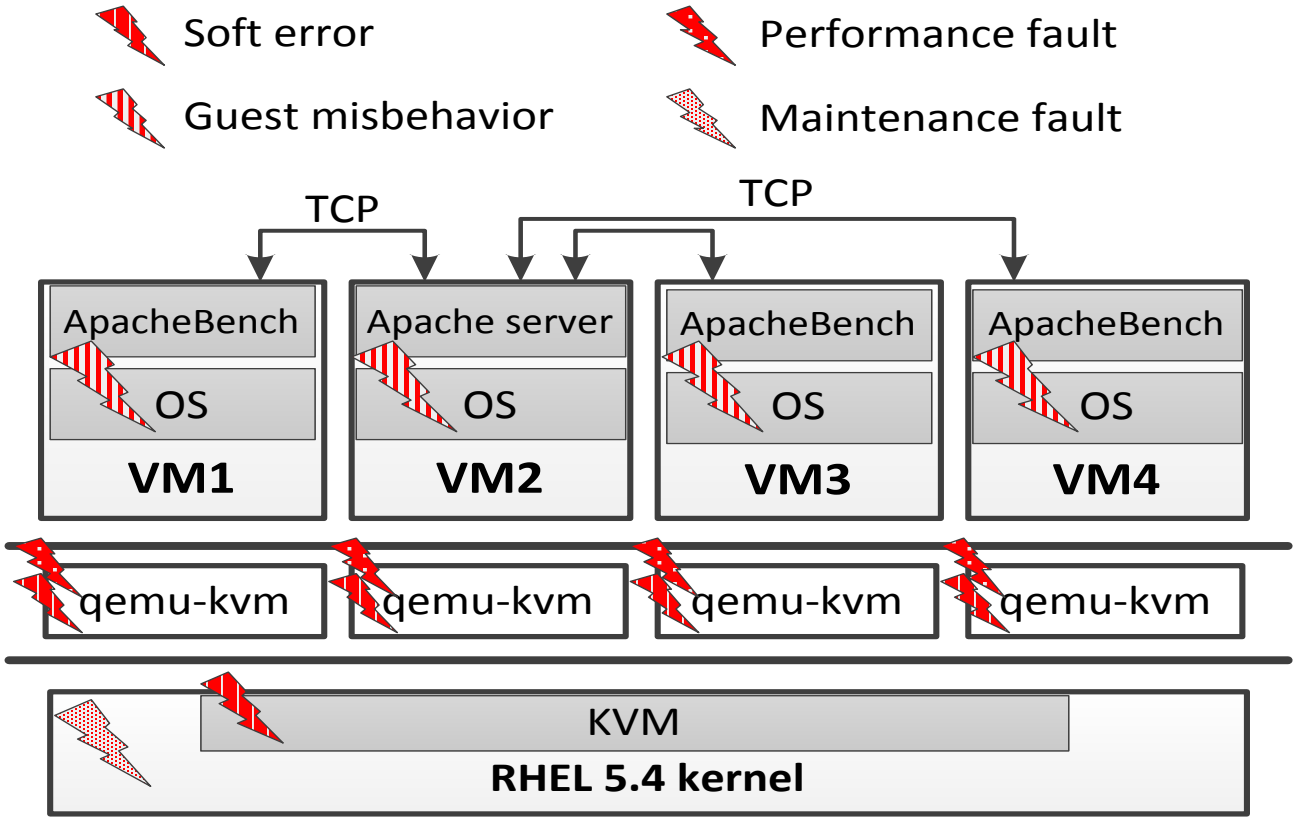


- Hypervisor: KVM (Xen – not depicted)
- Management system:
  - Virt-manager (http://virt-manager.et.redhat.com/)

Identify fault models



Experiment Set up



- ApacheBench VMs create connections and send HTTP GET requests to HTTP server VM.
- HTTP server VM accepts connections and replies requested pages

Experimental Results

Fault type	Hyper-visor	Target	Activated/Injected faults	Guest behavior	Hypervisor behavior
Guest misbehavior	**KVM**	**Code segment, data segment, stack segment and register of GUEST OS KERNEL**	**14,000 injected faults**	**Guest kernel crashes**	**No fault manifestation from guest to host is observed**
	Xen	XEN Data from [3]: Found two cases that the failures manifested from the guest to the host			
Turn one CPU core ON	**KVM**	Physical CPU core	10/10	N/A	10 kernel crashes
		CPU core in guest VM	10/10	No change	No Change
	Xen	CPU core in Dom0	10/10	No change	No change
		CPU core in guest VM	10/10	Guest kernel returns error, but executes normally	N/A
Soft errors (user space)	**KVM**	Data segment, stack segment and register of QEMU-KVM PROCESS	496/500 faults	N/A	120 qemu-kvm crashes 26 qemu-kvm defunct (zombie) state No Kernel crashes
	Xen	qemu-dm	100/100	Guest VM stopped when qemu-dm crashed	55 qemu-dm crashes. 12 defunct (zombie) state. No kernel crash.
		xenstored	100/100	No change	No change

Experimental Results

Fault type	Hyper-visor	Target	Activated/Injected faults	Guest behavior	Hypervisor behavior
Guest misbehavior	**KVM**	Code segment, data segment, stack segment and register of GUEST OS KERNEL	14,000 injected faults	Guest kernel crashes	No fault manifestation from guest to host is observed
	Xen	XEN Data from [3]: Found two cases that the failures manifested from the guest to the host			
Turn one CPU core ON	**KVM**	**Physical CPU core**	**10/10**	**N/A**	**10 kernel crashes**
		CPU core in guest VM	10/10	No change	No Change
	Xen	CPU core in Dom0	10/10	No change	No change
		CPU core in guest VM	**10/10**	**Guest kernel returns error, but executes normally**	**N/A**
Soft errors (user space)	**KVM**	Data segment, stack segment and register of QEMU-KVM PROCESS	496/500 faults	N/A	120 qemu-kvm crashes 26 qemu-kvm defunct (zombie) state No Kernel crashes
	Xen	qemu-dm	100/100	Guest VM stopped when qemu-dm crashed	55 qemu-dm crashes. 12 defunct (zombie) state. No kernel crash.
		xenstored	100/100	No change	No change

Experimental Results (2)

Fault type	Hyper-visor	Target	# Activated/Injected faults	Guest behavior	Hypervisor behavior
Soft errors (kernel space)	**KVM**	Data segment, code segment and register of KVM KERNEL MODULE	380/1000	N/A	94 kernel crashes
	Xen	Crashing Dom0 by inserting a faulty kernel module	20/20	All guest VMs stopped	20 kernel crashes
Performance faults	**KVM**	**Threads in QEMU-KVM PROCESS**	**399/400 faults**	**DoS during injected fault**	**16 kernel crashes**
	Xen	N/A	N/A	N/A	N/A

Fault Analysis: Multi-threads delay

Run No.	Trigger point	# injected thread	Delay time	Duration time	Activated/Injected faults	Number of kernel crashes
1	Sampled addresses	1, 2, 3, or 4	rand(100) seconds	rand(200) seconds	399/400	16
2	Only one address that crashed the kernel in run 1	1 or 2	rand(100) seconds	rand(100) seconds	80/80	0
3	Same address as run 2	3 or 4	rand(100) seconds	rand(100) seconds	100/100	3

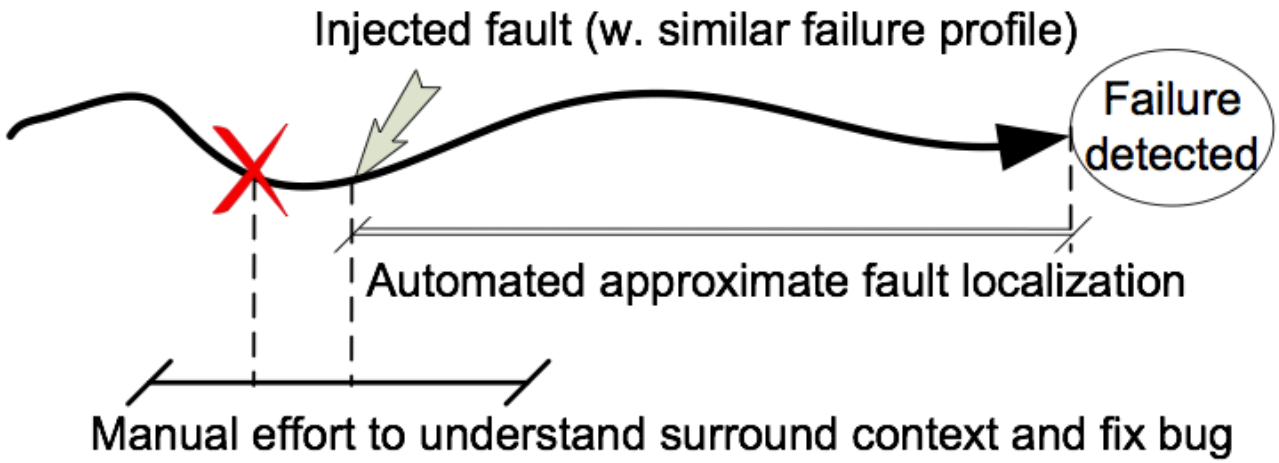
- Observation:
  - Non-deterministic behavior
    - kernel crashes only when injecting delay to 3-4 threads
  - Kernel call traces show the involvement of KVM module
- Potential cause: Race condition in KVM module

Summary

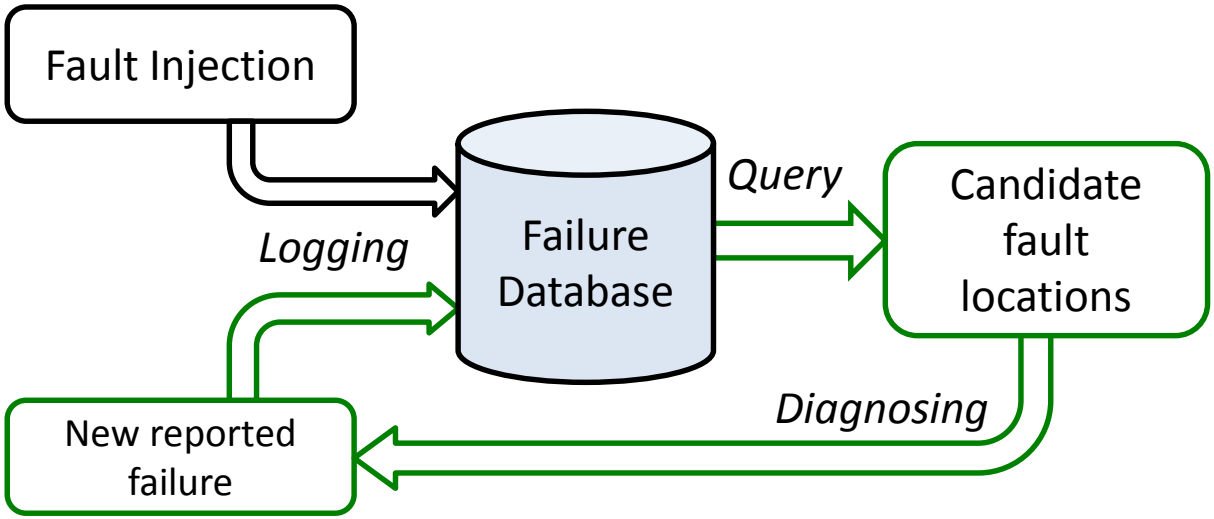
- Fault injection based evaluation of VM environments
  - failure isolation mechanisms, maintainability, and completeness of the implementation of the two hypervisors and the management system
  - fault/error models: guest system misbehavior, soft errors, performance faults, and maintenance faults
- Key results
  - **virtual machine ≠ physical machine:** a guest system may behave differently in different virtualization environments
  - Turning on CPUs/cores my lead to the hypervisor failure (e.g., crash of KVM-hypervisor when turning on a CPU core)
  - Bugs in the hot-plugging mechanism of the virtual CPU (e.g., unable to turn on the virtual CPU in Xen) can make system vulnerable to malicious attacks

Failure Diagnosis using Message Flow Reconstruction and Targeted Fault Injection

Approximate Fault Localization: Concept

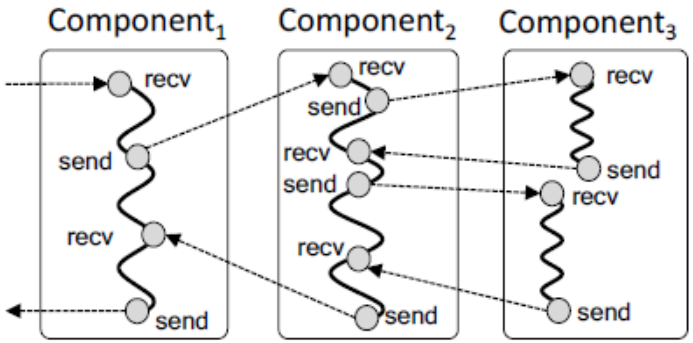


Assumption: **Similar failure profiles imply similar faults**



Approximate Fault Localization: Approach

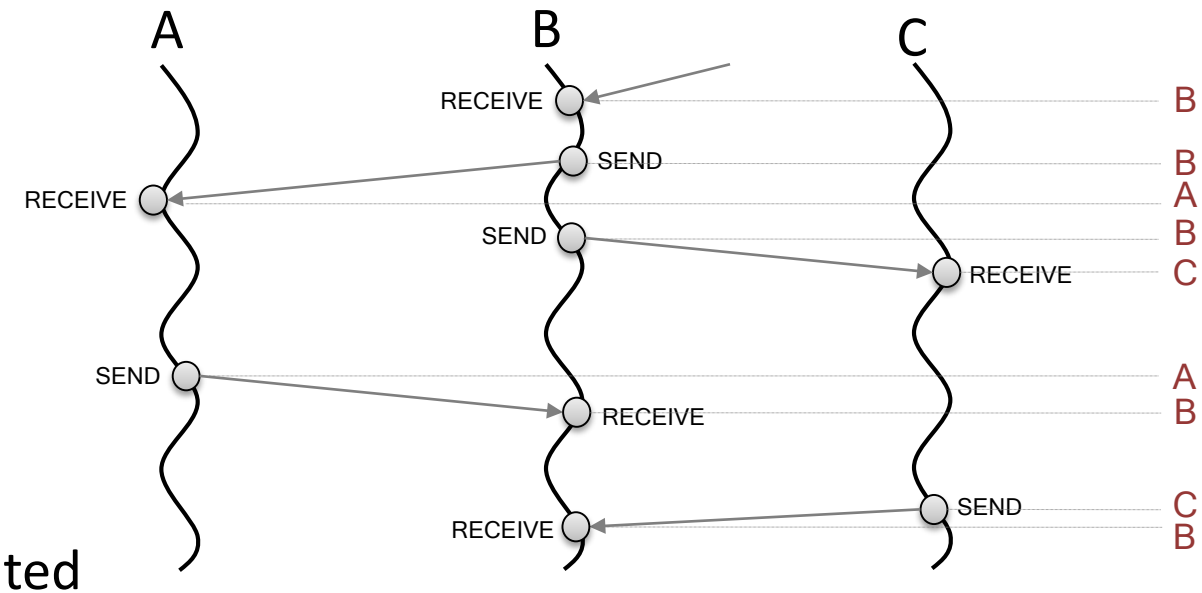
- Upon a failure in a system collect a failure profile
  - e.g. in terms of *sent* and *received* messages
- Process failure profile to reconstruct an end-to-end processing flow corresponding to the failure



  - a sequence of system events across distributed components invoked to process a user/application request
- Use the reconstructed processing flow to query against a pre-constructed failure profiles stored in Failure Profiles Database
  - Use “string edit distance” metric to identify similar flows and “pinpoint” the fault location

Message Flow Reconstruction and Comparison

- Need to represent event flows so to enable fast identification of similar flows
- Event flows translated into event strings
  - an event in a string represented as a letter that corresponds to the source component of this event, e.g., ***BBABCABCB***
  - event order based on timestamps
- Compare flows using *String Edit Distance*
  - the minimum number of insertion, deletion, or replacement of a letter required for changing one string into the other



Enabling Techniques

- Distributed Events Tracing

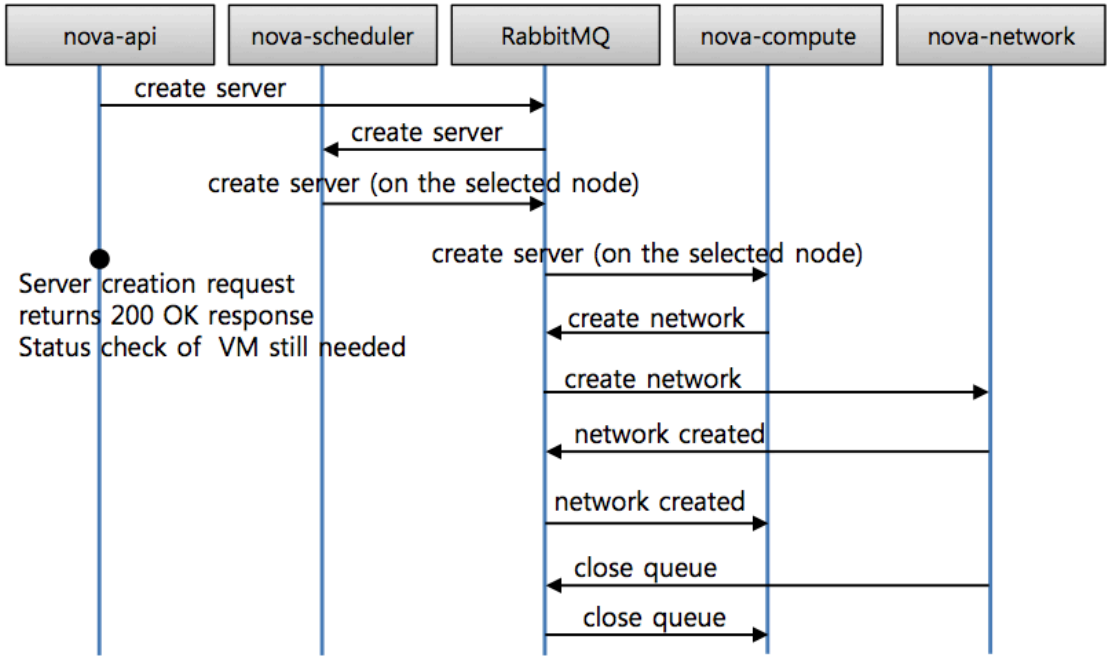
  record system events (e.g., syscall, library call) in distributed systems

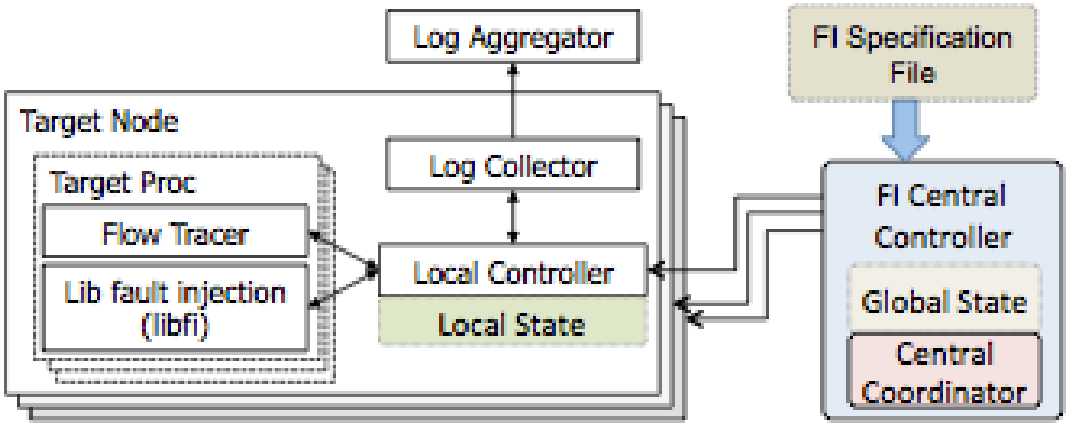
- Message Flow Reconstruction and Comparison

  quantify the dissimilarity between failure profiles

- Targeted Fault Injection

  deterministically inject faults at exact locations in the execution flow of a distributed system





Evaluation

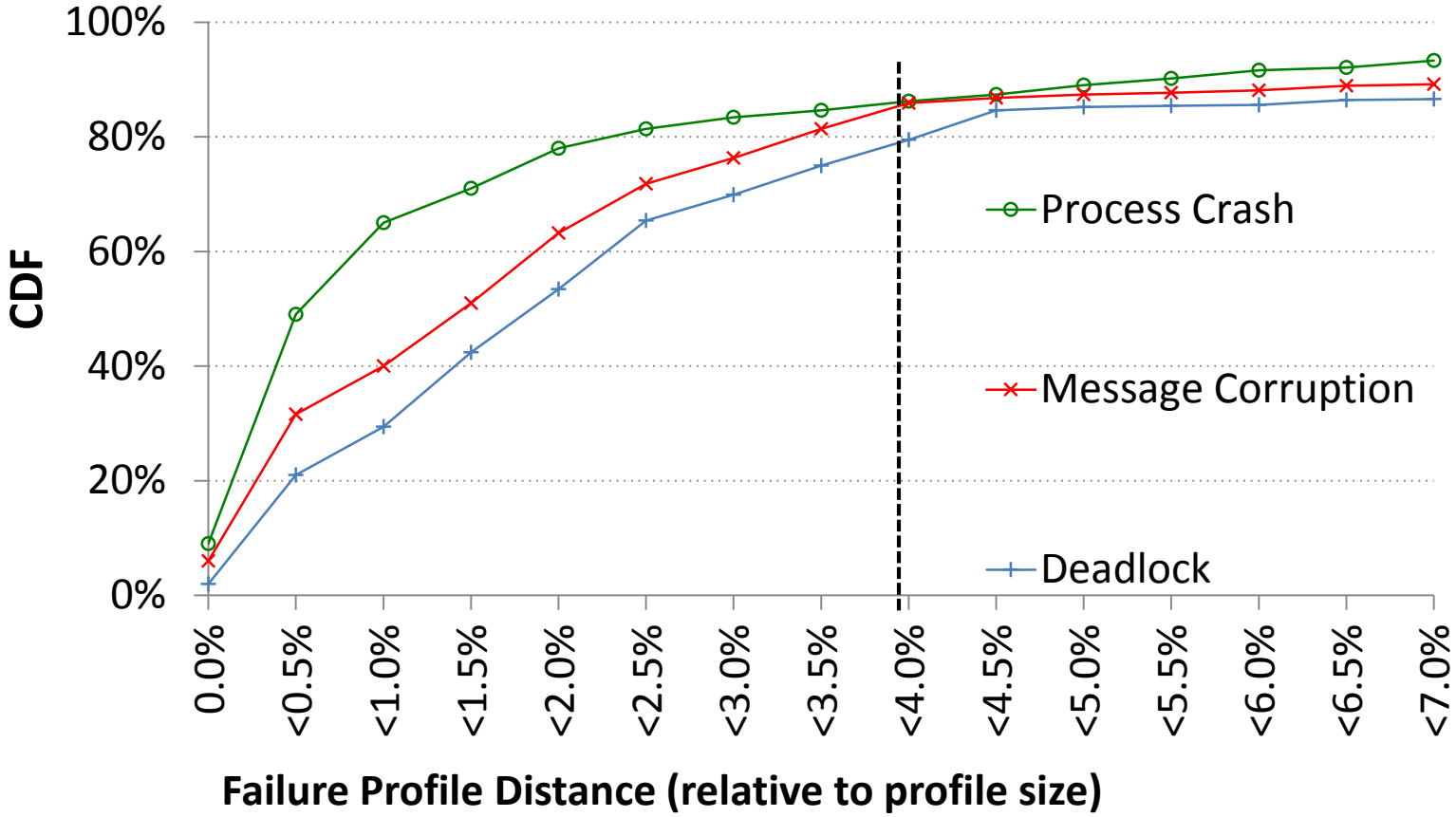
- ***Target:*** OpenStack, an open source distributed cloud management system
- ***Validation***
  - Do similar failure profiles imply similar faults?
- ***Evaluation of AFL Accuracy***
  - Identification of fault type and affected component(s)
  - *Fault distance* between the determined fault location and the actual injected fault (Top-K nearest faults)
    - fault distance measured as the number of *libc* calls between the determined approximate fault location and the actual fault location in the end-to-end flow of fault-free execution

Construction of Failure Profile Database (FPDB)

Fault Type	Location Type	$\lvert\mathbb{F}\rvert$	$\lvert\mathbb{FP}\rvert$
Process Crash	All monitored libc calls	23323	116589
Message Corruption	All *read*, *write*, *send*, and *recv* libc calls	18221	91092
Deadlock	All thread and lock related libc calls	2143	10702

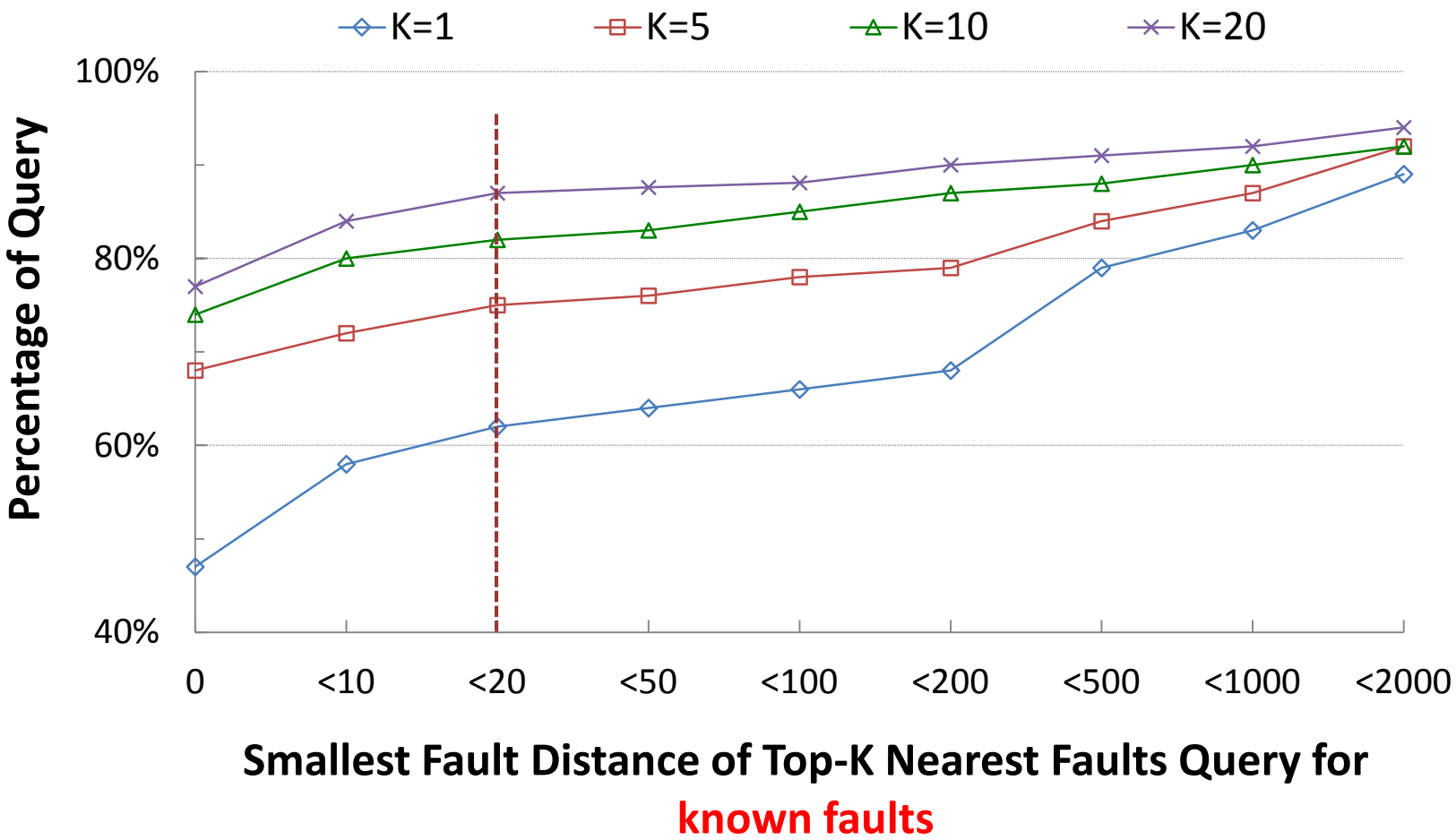
- The FPDB is constructed for VM Provision (nova boot) requests
- Five failure profiles collected for each fault
- Fault models:
  - Contained faults: *Process crash*, *deadlock* (within a process)
  - Propagated faults: *Message corruption*

Do Similar Failure Profiles Imply Same faults?



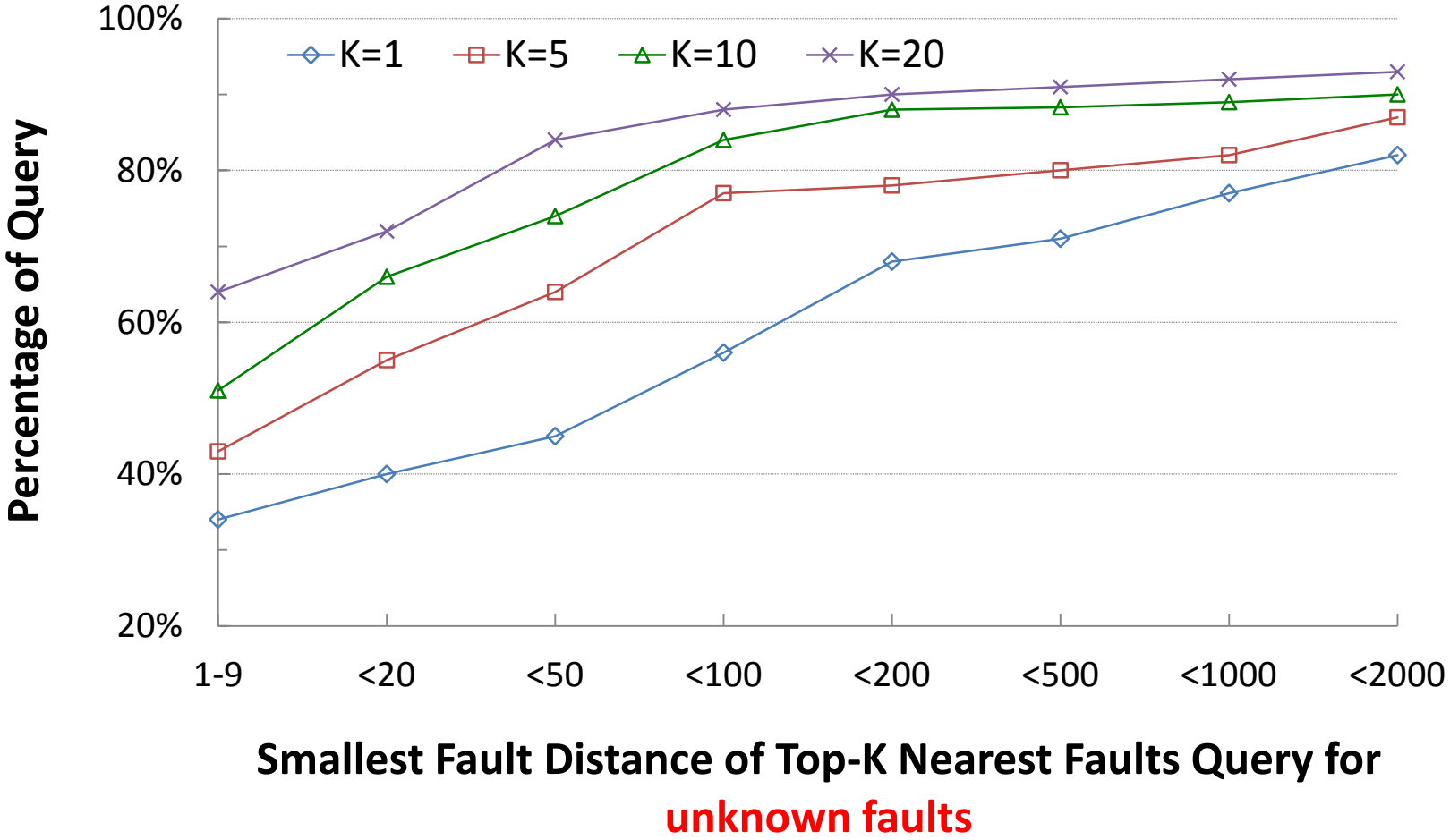
More than 80% of all the injected faults, across all three fault models, result in less than 4% of the failure profile variation

Accuracy of AFL: Top-K Nearest Faults for Known Faults



50% of the Top-1 query results contain the exact fault locations, i.e., fault distance is zero

Accuracy of AFL: Top-K Nearest Faults for Unknown Faults



Two orders of magnitude better than OpenStack's error reporting mechanism

Evaluation of FPDB against Real Bugs

- Use information on bug fixes included in the OpenStack Grizzly 2013.1 released in April 2013.
- Reproduce each bug and capture the bug traces in testing environment
  - 14 bugs (out of 766) selected
- information (from bug reports) on bug locations as the *ground truth* to compare against the locations returned from querying FPDB

- Measures: use Top-10 queries to find injections that generated closest traces to each reproduced trace of selected bugs
- Compute fault distance (in terms of the number of LibC function invocations) between each returned injected location and the locations where the code was fixed for each bug

Query Results

- Reproduced six VM provision-, four VM resize-, and four VM migration-related bugs
- 8 out of 14 queries returned at least one trace that had fault distance within 100 LibC invocations
- Identified fault-to-failure propagation paths provided useful indicators for locating and fixing actual bugs

Request type	VM Provision						VM Resize				VM Migration			
Smallest fault distance (in LibC calls)	**2**	14	**18**	221	236	565	**14**	56	84	245	**34**	45	342	442

Example of Real Bug Report

LibvirtBridgeDriver crashes when spawning an instance with NoopFirewallDriver

Bug #1050433 reported by [redacted] on 2012-09-13

Bug Description

```
I am trying the LibvirtHybridOVSBridgeDriver, with the uptodate
devsatck and the vlan manager.
```

Required system configuration

```
I've created one network for the demo project :
nova-manage network create net2 --fixed_range_v4=172.16.2.0/24 --
num_networks=1 --network_size=256 --vlan=1001

I boot an instance :
nova boot --flavor 1 --image 000f2e5b-e1d1-4bc0-a9d1-3e07d527f7f1 vm_1
```

How to reproduce the bug

```
the instance can't be launched du to the following error :

2012-09-13 15:14:06 ERROR nova.compute.manager [req-a10f2d0d-4992-4b50-
a297-b7b188ab98a1 demo demo] [instance: b6fab81d-d700-4a4a-
bb3f-88422e3bc40f] Instance failed to spawn
…
```

OpenStack error messages

Example of Localizing a Real Bug

- **Reported Bug:** LibvirtBridgeDriver crashed when spawning an instance with NoopFirewallDriver
- ***Localization using FPDB:*** closest failure profile to that bug generated by message corruption fault to ***write()*** LibC function invoked by *nova-compute process*.
  - write() sends network filter configuration from the nova-compute to the socket of libvirtd (executs VM provisions)
- Injected fault corrupted the name of the network filter

```
Original value:
nova-compute 7fad40b08700  6208   write (15, AF_FILE
    =>/var/run/libvirt/libvirt-sock) = 74: <filter
    name='nova-instance-instance-ID-MACADDRESS'
    chain='root'>

New value (the 33rd character is corrupted):
nova-compute 7fad40b08700  6208   write (15, AF_FILE
    =>/var/run/libvirt/libvirt-sock) = 74: <filter
    name='nova-instance-instbnce-ID-MACADDRESS'
    chain='root'>
```

Example of Localizing a Real Bug

- Tracing back data flow of that message content in the execution of nova-compute found that
  - configuration for a network filter is generated regardless of whether the NoopFirewallDriver option is selected
  - implementation of the NoopFirewall driver is not intended to recognize a network filter configuration
  - an exception prevents VM provision

- Network configuration function had to be fixed to validate the NoopFirewallDriver option before generating a network filter configuration

Limitations

- Not effective for failures that do not alter the processing flow, e.g., a bug that only causes a performance degradation
- No weights assigned to different parts of the processing flow.
- Developers need to select the right answer based on the ranking (trace distance) provided by the localization method.
- Approximate bug locations indicate a context, in which a bug might occur
- Overhead of tracing in production systems

Summary

- Develop low-cost method for the approximate fault localization
  - reduce the cost of fault diagnostic while providing precision close to the methods used for the exact fault localization
  - support large complex distributed environments such as the cloud computing
- Demonstrate effectiveness of the prototype implementation on the OpenStack
  - effective in determining (approximate) fault/error locations
  - accurate in identifying the failure types and the affected components

Recommended Readings

D. Stott et al., “NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors,” Computer Performance and Dependability Symposium, 2000. IPDS 2000. Proceedings. IEEE International , vol., no., pp.91-100, 2000.

M. Le, A. Gallagher, and Y. Tamir, “Challenges and opportunities with fault injection in virtualized systems,” In Proceedings of the 1st Int. Workshop on Virtualization Performance: Analysis, Characterization, and Tools, Apr. 2008.

A. Kivity et al., “kvm: the Linux virtual machine monitor,” In OLS '07: The 2007 Ottawa Linux Symposium, pages 225--230, July 2007.

Z. Mwaikambo et al., “Linux kernel hotplug CPU support,” In Proceedings of the Linux Symposium, Vol. 2, 467--480.

T. Tsa et al., “Stress-Based and Path-Based Fault Injection,” IEEE Trans. Computers, vol. 48, no. 11, pp. 1183-1201, Nov. 1999.

M. Armbrust et al., “A view of cloud computing,” Commun. ACM, vol. 53, Number 4 (2010), 50-58.

A. Stern, “Update from Amazon Regarding Friday's S3 Downtime,” CenterNetworks (Feb. 2008).

S. Wilson, “AppEngine Outage,” CIO Weblog (June 2008).

WindowsAzure, “The Windows Azure Malfunction This Weekend,” MSDN Blog (Mar 2009).

http://virt-manager.et.redhat.com/. Virt-manager

M. Hsueh, T.K. Tsai, and R.K. Iyer, “Fault Injection Techniques and Tools,” Computer , vol.30, no.4, pp.75-82, Apr 1997.

D. Siewiorek, R. Chillarege and Z. Kalbarczyk, “Reflections on industry trends and experimental research in dependability,” IEEE Transactions on Dependable and Secure Computing, vol.1, no.2, 2004.

D. Chen et al., "Error Behavior Comparison of Multiple Computing Systems: A Case Study Using Linux on Pentium, Solaris on SPARC, and AIX on POWER," 14th IEEE Pacific Rim International Symposium on Dependable Computing, 2008,

J. Carreira; H. Madeira and J.G. Silva; "Xception: a technique for the experimental evaluation of dependability in modern computers," Software Engineering, IEEE Transactions on, vol.24, no.2, pp.125-136, Feb 1998.

W. Gu et al., "Characterization of Linux Kernel Behavior under Errors," Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on, vol., no., pp. 459- 468, 22-25 June 2003.

A. Li et al., “CloudCmp: shopping for a cloud made easy,” In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (HotCloud'10). USENIX Association, 2010.

T. Ristenpart et al. “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds,” In Proceedings of the 16th ACM conference on Computer and communications security (CCS '09). ACM, New York, USA,

Paul Barham et al. “Xen and the art of virtualization,” SIGOPS Oper. Syst. Rev. 37, 5 (October 2003), 164-177.

Recommended Readings

S. K. Sahoo, J. Criswell, C. Geigle, and V. Adve, “Using likely invariants for automated software fault localization,” in Proc. of the 8th Intl Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York, NY, USA: ACM, 2013, pp. 139–152.

R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, “On the accuracy of spectrum-based fault localization,” in Proc. of the Academic and IndustrialConf. on Testing: Practice and Research Techniques - MUTATION. Washington, DC, USA: IEEE Computer Society, 2007, pp. 89–98.

C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff, “Statistical debugging: A hypothesis testing-based approach,” IEEE Transactions on Software Engineering, vol. 32, no. 10, pp. 831–848, 2006.

V. Dallmeier, C. Lindig, and A. Zeller, “Lightweight defect localization for Java,” in ECOOP 2005-Object-Oriented Programming. Springer, 2005, C. Liu, X. Yan, H. Yu, J. Han, and S. Y. Philip, “Mining behavior graphs for “backtrace” of noncrashing bugs.” in SDM. SIAM, 2005, pp. 286–297.

B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang, “vPath: Precise discovery of request processing paths from black-box observations of thread and network activities,” in Proc. of the USENIX Annual Technical Conf., Berkeley, CA, USA, 2009.

B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, “Dapper, a large-scale distributed systems tracing infrastructure,” Google, Inc., Tech. Rep., 2010. [Online]. Available: http://research.google.com/archive/papers/dapper-2010-1.pdf

Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel, “Rebucket: A method for clustering duplicate crash reports based on call stack similarity,” in Proc. of the 2012 Intl. Conf. on Software Engineering. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1084–1093.

K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt, “Debugging in the (very) large: Ten years of implementation and experience,” in Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles. New York, NY, USA: ACM, 2009

C. Liu and J. Han, “Failure proximity: A fault localization-based approach,” in Proc. of the 14th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering. ACM, 2006, pp. 46–56.

A. Anandkumar, C. Bisdikian, and D. Agrawal, “Tracking in a spaghetti bowl: Monitoring transactions using footprints,” in Proc. of the 2008 ACM SIGMETRICS Intl. Conf. on Measurement and modeling of Computer Systems (SIGMETRICS ’08). New York, NY, USA: ACM, 2008, pp. 133–144.

R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, “X-trace: A pervasive network tracing framework,” in Proc. of the 4th USENIX Conf. on Networked Systems Design & Implementation (NSDI’07). Berkeley, CA, USA: USENIX Association, 2007, pp. 20–20.

H. S. Gunawi, T. Do, J. M. Hellerstein, I. Stoica, D. Borthakur, and J. Robbins, “Failure as a service (FaaS): A cloud service for large-scale, online failure drills,” EECS Department, University of California, Berkeley, Tech. Rep., 2011. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-87.pdf

R. Chandra, R. M. Lefever, K. R. Joshi, M. Cukier, and W. H. Sanders, “A global-state-triggered fault injector for distributed system evaluation,” IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 7, pp. 593–605, 2004.

X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva, “On fault resilience of OpenStack,” in Proc. of the 4th Annual Symp. on Cloud Computing. ACM, 2013.

W. E. Wong and V. Debroy, “A survey of software fault localization,” University of Texas at Dallas, Tech. Rep. UTDCS-45-09, 2009.

I. Lee and R. Iyer, “Diagnosing rediscovered software problems using symptoms,” IEEE Transactions on Software Engineering, vol. 26, no. 2, pp. 113–127, 2000.