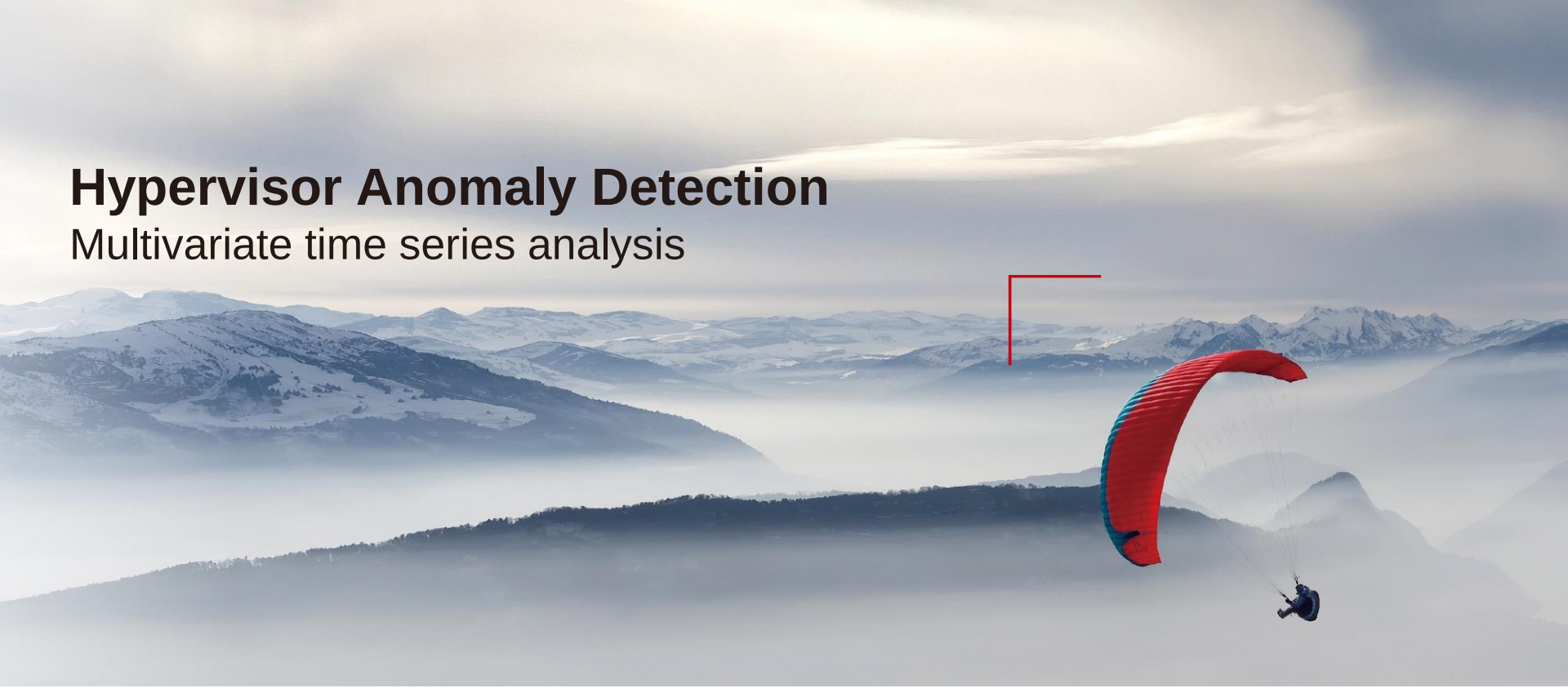
Hypervisor Anomaly Detection

Multivariate time series analysis



Lecture at Technical University of Berlin

Jorge Cardoso
Chief Engineer for Hyperscale AIOps
Munich Research Center

2022.04.12

Technische Universität Berlin

HUAWEI

Indirect Hypervisor Anomaly Detection

Description

Background. Hypervisors create and manage virtual machines (VMs). As public clouds become critical infrastructures, the reliability of hypervisors and VMs becomes increasingly important.

Problem. However, the reliability of the virtualization layer has not been properly addressed and mechanisms to evaluate its health are limited. For example, if a hypervisor faces a fail-stop or gray failure, the failure is propagated to all hosted virtual machines. This makes production VMs dependent on a single point of failure.

Objective. The objective of this research is to detect anomalous hypervisors based on the behavior of the VMs they manage.

Approach. Thus, to identify hypervisors that are experiencing failures, we developed the HAD (Hypervisor Anomaly Detection) algorithm. The objective is to preemptively migrate VMs to another hypervisor when the underlying hypervisor is faulty. HAD design was inspired on Attribution Theory. HAD observes and learns the behavior of metrics of individual virtual machine running on a hypervisor. When a VM exhibits a different behavior, HAD determines if the behavior is intrinsic to the VM or it is observed across several VMs. Finally, the algorithm attributes the observed behavior to either internal causes (e.g., a VM is stressed because it is running a CPU intensive application) or external causes (e.g., a hypervisor failure).

Results. HAD is an online algorithm which achieves up to 14x speed improvement when compared to its offline implementation. It implements a new online algorithm for root-cause attribution. It was evaluated with metrics collected from VMs running on Azure cloud and an F1 score of >90% was achieved.

Indirect Hypervisor Anomaly Detection

Detecting Faulty Hypervisors using VMs Metrics

Virtualization failures affect VMs but cannot be observed directly

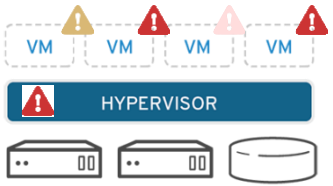


Fig. VM exhibit abnormal behavior when the hypervisor has technical issues

Problem

- Hypervisors do not have effective methods to determine their health

Indirect approach to detect hypervisor failures by monitoring VMs

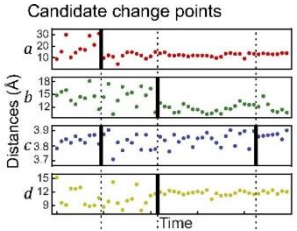


Fig. Several time-series generated by several VMs running in the same hypervisor

- **Insight**. Resource saturation of VMs suddenly changes, within a window w, when an hypervisor is malfunctioning

MAIN ACHIEVEMENT

HAD algorithm – quorum change-point detection

- Analyzes individual time-series, and uses change points and voting to decide whether there is an hypervisor malfunction
- Key results: **F1 score 72%** (data from 2 VMs); **80+%** (3+ VMs)

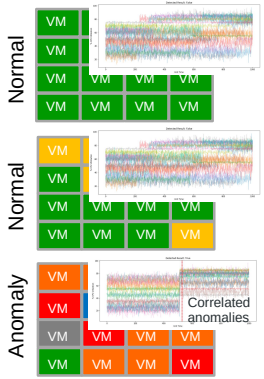
HOW IT WORKS

Method 1 (Change Points)
1. Treat time-series as univariate
2. Detect change points
3. Vote to decide global changes

Method 2 (Isolation Forest)
1. Treat time-series as features
2. Detect significant changes

Method 3 (ECP E.Divisive)
1. Treat time-series as multivariate
2. Detect multiple change points

Analyze VM resources to detect correlated anomalies



ASSUMPTIONS & LIMITATIONS

- Datasets used for evaluation were collected from simulation environment, synthetic data generator and public sources

TRL 5. *Basic technological components are integrated with realistic supporting elements so it can be evaluated in test environment*

Predictive Maintenance for Hypervisors

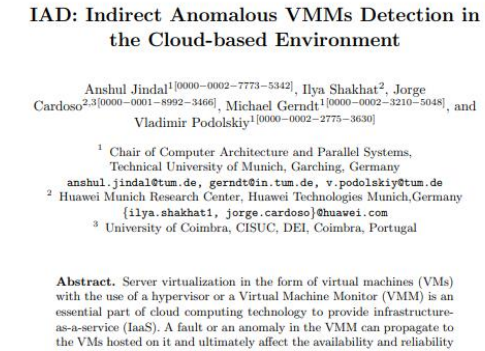
Migrate customers' VMs before hypervisors completely fail

Results on Synthetic Data

Sr.No.	Algorithm	F1-Score	Time Taken	Acc.	Precision	Recall	TP	FP	TN	FN	TTP	TTN
1	IAD_v1(10,5)	0	27.63	0.5	0	0	0	0	5	5	5	5
2	IAD_v1(20,5)	0.57	28.33	0.7	1	0.4	2	0	5	3	5	5
3	IAD_v1(30,5)	0.75	29.18	0.8	1	0.6	3	0	5	2	5	5
4	IAD_v1(40,5)	0.75	30	0.8	1	0.6	3	0	5	2	5	5
5	IAD_v1(50,5)	0.89	30.94	0.9	1	0.8	4	0	5	1	5	5
6	IAD_v1(60,5)	0.89	32.28	0.9	1	0.8	4	0	5	1	5	5

Benefits

- Reduce the number of VMs crashes due to hypervisor failures in 80%



IAD: Indirect Anomalous VMMs Detection in the Cloud-based Environment

Anshul Jindal[1][0000−0002−7773−5342], Ilya Shakhat[2], Jorge Cardoso[2,3][0000−0001−8992−3466], Michael Gerndt[1][0000−0002−3210−5048], and Vladimir Podolskiy[1][0000−0002−2775−3630]

[1] Chair of Computer Architecture and Parallel Systems, Technical University of Munich, Garching, Germany
anshul.jindal@tum.de, gerndt@in.tum.de, v.podolskiy@tum.de
[2] Huawei Munich Research Center, Huawei Technologies Munich,Germany
{ilya.shakhat1, jorge.cardoso}@huawei.com
[3] University of Coimbra, CISUC, DEI, Coimbra, Portugal

Abstract. Server virtualization in the form of virtual machines (VMs) with the use of a hypervisor or a Virtual Machine Monitor (VMM) is an essential part of cloud computing technology to provide infrastructure-as-a-service (IaaS). A fault or an anomaly in the VMM can propagate to the VMs hosted on it and ultimately affect the availability and reliability

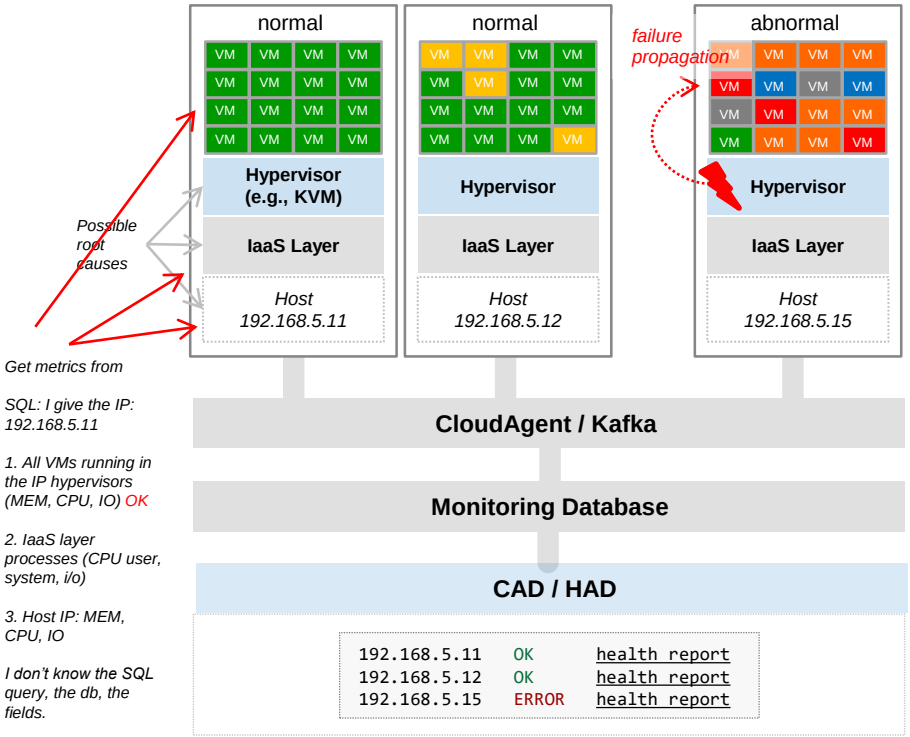
IAD: Indirect Anomalous VMMs Detection in the Cloud-based Environment. Jindal, A.; et al.. AIOPS 2020 International Workshop on Artificial Intelligence for IT Operations, Springer, 2021

Problem Definition
Limited Health Monitoring

Background
- Hypervisors manage virtual machines (VMs)
- As public clouds become critical infrastructures, the reliability of hypervisors becomes increasingly important

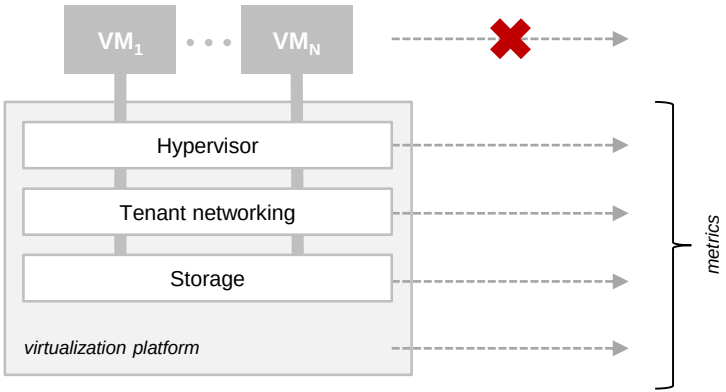
Problem
- For example, if a hypervisor faces a fail-stop or gray failure, the failure is propagated to all hosted virtual machines
- This makes production VMs dependent on a single point of failure.

Approach
- Existing mechanisms to evaluate the health of hypervisors is limited
- Observes and learns the behavior of metrics of individual VM running on a hypervisor
  - (next slide discuss the type of behavior monitored)
- When VMs exhibit different behaviors, determine if the behavior is intrinsic to VMs or it is observed across several VMs



Problem overview

Compute service



Monitoring

VM monitoring is not available

- VMs belong to customers and do not have monitoring agents inside

Platform monitoring

- BM host metrics:
  - CPU, network IO, disk IO
- per-VM metrics:
  - total CPU utilization
  - total memory allocation (virtual memory assigned to VM)
  - network IO (packet transmitted and received, error count) per network card
  - storage IO (bytes read and written, timings) per volume attached to VM

Problem statement

Failures of virtualization platform affect VMs but could not be observed directly.

Example 1: network storage throughput is affected by failure, causing latency in all disk operations in all VMs

Example 2: tenant networking is affected by packet loss in underlying vxlan tunnel, it cannot be measured directly on tunnel level, but affects TCP traffic between VMs.

Proposed solution

By knowing the model of VMs' behavior, check if it persists over the time. Simultaneous change for all VMs may indicate problems in the virtualization platform.

Challenges

- Type of VM workload is not known, it may even change during VM lifetime
- Many VMs could be in almost idle state
- VMs change over the time (some come, some go)

Hypervisor Health
Behavior Evaluation

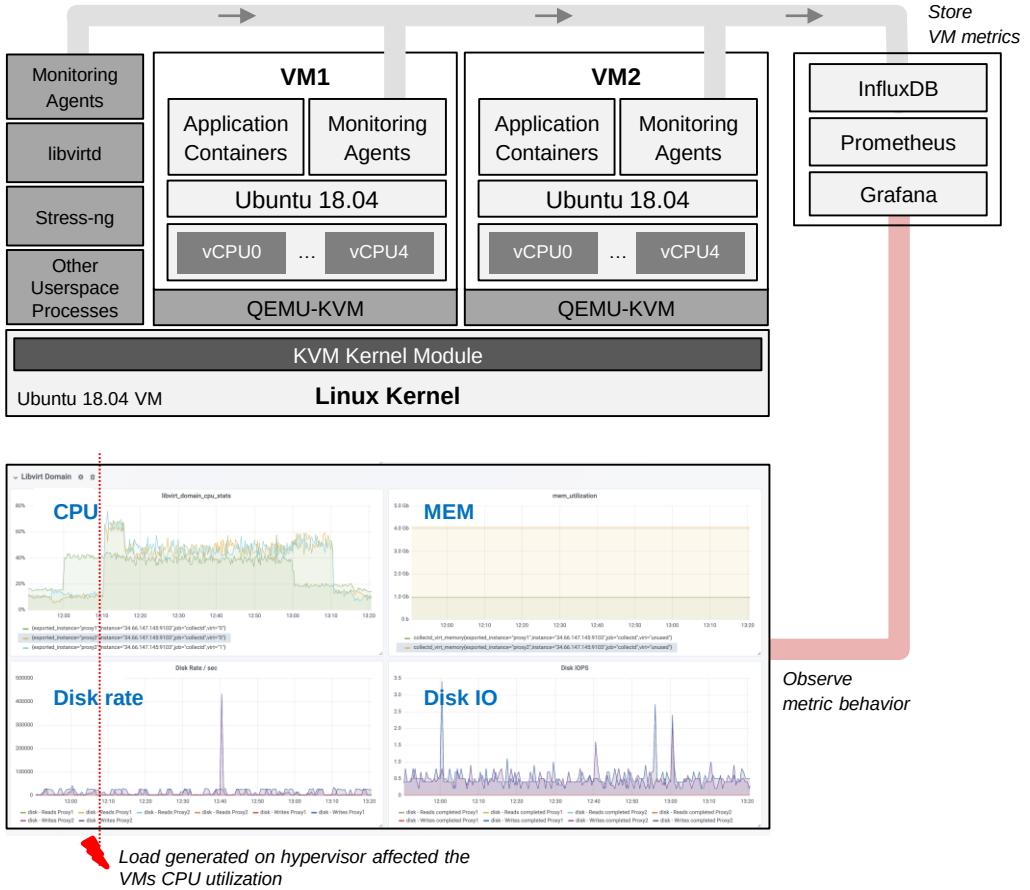
Our experiments indicate that the best way to detect the failure of hypervisors is to monitor the **CPU usage** of VMs

Observation

- We analyzed the effect of overloading the host running the hypervisor
- 4 metrics were monitored: CPU, MEM, Disk Rate and IO
- CPU metric is the most affected and visualized parameters in the VMs
- CPU in the Virtual machines decreases if there is certain workload going on them
- CPU in the virtual machines increases if there is no workload going on them
- All or most of the virtual machines are affected when a load is introduced on hypervisor

Further research

- How generic and representative is the fault (i.e., hypervisor load)?



KVM

KVM (Kernel-based Virtual Machine) hypervisor is a kernel module (kvm.ko). The process that runs KVM is typically the QEMU (Quick Emulator) process.

KVM/QEMU combination works as follows:

- **KVM Module**: Loaded into the Linux kernel. It provides the necessary infrastructure for hardware-assisted virtualization.

- **QEMU Process**: User-space emulator that interfaces with the KVM kernel module for creating and managing virtual machines.

When starting a virtual machine using qemu-system-x86_64, it launches a QEMU process that calls KVM module:

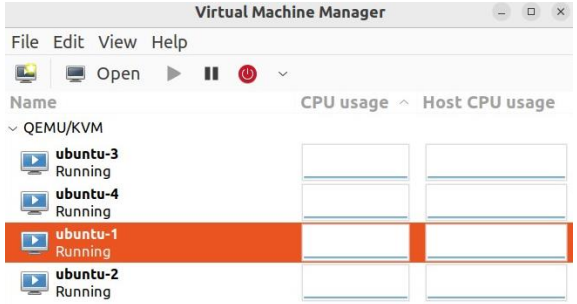
qemu-system-x86_64 -enable-kvm -cpu host -m 2048 -hda your_disk_image.qcow2

- -enable-kvm: Enables KVM for hardware virtualization.
- -cpu host: Uses the host CPU features.
- -m 2048: Allocates 2 GB of RAM for the virtual machine.
- -hda your_disk_image.qcow2: Specifies the disk image for the virtual machine

Tools such as libvirt or virt-manager abstract the details of starting KVM/QEMU virtual machines.

```
$ ps -ef | grep qemu-system-x86_64
libvirt+  756677       1  0 Dez14 ?        00:14:21
/usr/bin/qemu-system-x86_64 -name guest=ubuntu-
1,debug-threads=on -S -object {"qom-
type":"secret","id":"masterKey0","format":"raw","fi
le":"/var/lib/libvirt/qemu/domain-50-ubuntu-
1/master-key.aes"} -machine pc-q35-
6.2,usb=off,vmport=off,dump-guest-core=off,memory-
backend=pc.ram -accel kvm -cpu host,migratable=on -
m 2048 -object {"qom-type":"memory-backend-
ram","id":"pc.ram","size":2147483648} -overcommit
mem-lock=off -smp 2,sockets=2,cores=1,threads=1 -
uuid 283f1e7c-c249-44b7-bcd5-b48e406c32e4 -no-user-
config -nodefaults -chardev
socket,id=charmonitor,fd=43,server=on,wait=off -mon
chardev=charmonitor,id=monitor,mode=control -rtc
base=utc,driftfix=slew -global kvm-
pit.lost_tick_policy=delay -no-hpet -no-shutdown -
global ICH9-LPC.disable_s3=1 -global ICH9- …
```

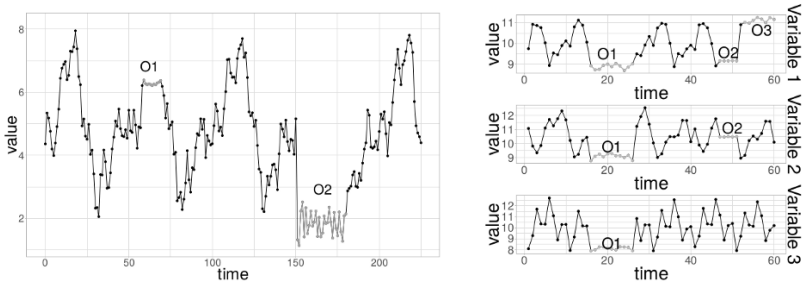
ps -ef | grep qemu-system-x86_64



https://virt-manager.org/

Theory
Anomaly detection for multivariate time series

Model	Paper reference
PCA	**[2003]** Shyu M L, Chen S C, Sarinnapakorn K, et al. A novel anomaly detection scheme based on principal component classifier
iForest	**[ICDM'2008]** Fei Tony Liu, Kai Ming Ting, Zhi-Hua Zhou: Isolation Forest
LODA	**[Machine Learning'2016]** Tomás Pevný. Loda: Lightweight online detector of anomalies
LSTM	**[KDD'2018]** Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, Tom Söderström. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding



(a) Univariate time series. (b) Multivariate time series.

A review on outlier/anomaly detection in time series data, Ane Blázquez-García, et al.

Lightweight Online Detector of Anomalies (LODA)
Given a set of data points, the algorithm randomly creates numerous sparse random projections, and then fits a 1-dimensional density model. Then, “surprise” is measured based on the density estimator (i.e., adaptive histogram estimator).

Comprehensive list of techniques: https://github.com/durgeshsamariya/awesome-outlier-detection-resources

Theory
Anomaly detection for multivariate time series

As the problem above clearly has zero Anomaly data, this unsupervised Anomaly detection problem will be treated as an outlier detection problem. There are several approaches to deal with this problem, which are as follows:

Density Based
- RKDE: Robust Kernel Density Estimation
- DBSCAN: Density-Based Spatial Clustering of Applications with Noise
- EGMM: Ensemble Gaussian Mixture Model

Neighbour or Distance Based
- LOF: Local Outlier Factor
- ABOD: kNN Angle-Based Outlier Detector
- ORCA: Outlier detection and Robust Clustering

Quantile Based
- OCSVM: One-class SVM
- SVDD: Support Vector Data Description

Projection Based
- IFOR: Isolation Forest
- LODA: Lightweight Online Detector of Anomalies

Comprehensive list of techniques: https://github.com/durgeshsamariya/awesome-outlier-detection-resources

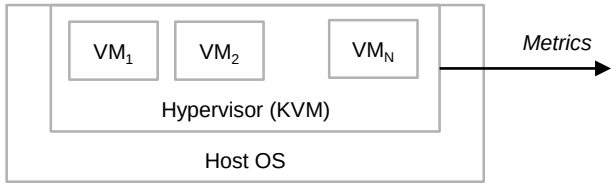
Metrics
Overview

Available metrics
- Metrics are collected from hypervisor via libvirt API
- List of available metrics:
  https://libvirt.org/html/libvirt-libvirt-domain.html#virConnectGetAllDomainStats
- Some metrics require guest agent to be installed. In this presentation only agent-less metrics are covered.

Metrics collection
- From the shell metrics can be viewed by command
  `virsh domstats <domain>`
- Collectd virt plugin.
- Direct Prometheus writer https://github.com/kumina/libvirt_exporter/

Nova metadata
- VMs created by OpenStack contain metadata with name, flavor and user information.
- Metadata can be retrieved by command:
  `virsh metadata <domain> --uri http://openstack.org/xmlns/libvirt/nova/1.0`
- Metadata retrieval is supported in collectd version 5.9 (possible to extend name with attributes extracted by XPath)

Metric retrieval



Collectd configuration:

```
LoadPlugin libvirt

<Plugin virt>
  Connection "qemu:///system"
  HostnameFormat name
  ExtraStats cpu_util disk operations disk_err
domain_state job_stats_background
job_stats_completed pcpu perf vcpupin
</Plugin>
```

Metrics
Perf events statistics

What is perf?

- Perf is a performance analyzing tool in Linux, and it can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing). Perf supports a list of measurable events, and can measure events coming from different sources. For instance, some event are pure kernel counters, in this case they are called software events, including context-switches, minor-faults, etc..

- Full list of performance events: https://libvirt.org/html/libvirt-libvirt-domain.html#virConnectGetAllDomainStats

How to:

- Get all perf events:
  `virsh perf <domain>`

- Enable perf event:
  `virsh perf <domain> --enable <event>`

- Get counter of the event:
  `virsh domstats <domain>`

- Disable event:
  `virsh perf <domain> --disable <event>`

Collectd configuration

- Collectd is a daemon which collects system and application performance metrics periodically and provides mechanisms to store the values in a variety of ways.
- Collecting perf metrics need to be enabled explicitly (https://collectd.org/documentation/manpages/collectd.conf.5.shtml#extrastats_string)
- All metrics are exported as *collectd_virt_perf_total* with event name specified as metric tag, e.g. *collectd_virt_perf_total{virt="perf_cpu_clock"}*

Metrics
CPU

Libvirt metric name	Unit	Collectd metric name	Description
cpu.time	nanosecond		Total CPU time spent for this domain
cpu.user	nanosecond	collectd_virt_ps_cputime_user_total	User CPU time spent
cpu.system	nanosecond	collectd_virt_ps_cputime_syst_total	System CPU time spent
vcpu.current	count		Current number of online virtual CPUs
vcpu.maximum	count		Maximum number of online virtual CPUs
vcpu.N.state	enum		State of the virtual CPU: 0 – offline, 1 – running, 2 – blocked (https://libvirt.org/html/libvirt-libvirt-domain.html#virVcpuState)
vcpu.N.time	nanosecond	collectd_virt_virt_vcpu_total{virt="N"}	Virtual CPU time spent by virtual CPU (https://libvirt.org/html/libvirt-libvirt-domain.html#virVcpuInfo)
vcpu.N.wait	nanosecond		Virtual CPU time spent by virtual CPU <num> waiting on I/O (https://www.redhat.com/archives/libvir-list/2015-December/msg00408.html)
	nanosecond	collectd_virt_virt_cpu_total_total	Total utilization of all guest virtual CPUs

It is expected that cpu.time = cpu.user + cpu.system + guest_time, where guest_time is sum of vcpu.N.time metrics

Metrics
Memory

Libvirt metric name	Unit	Collectd metric name	Description
balloon.current	kiB (1024 bytes)	collectd_virt_memory{virt="actual_balloon“}	The memory in kiB currently used (https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainInfo)
balloon.maximum	kiB	collectd_virt_memory{virt="total“}	The maximum memory in kiB allowed (https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainInfo)
balloon.last-update	second	collectd_virt_memory{virt="last_update“}	The time in seconds sine the UNIX epoch (1970-01-01) at which the statistics where last updated. 0 means that polling is not enabled.
balloon.rss	kiB	collectd_virt_memory{virt="rss“}	Resident Set Size of the running domain's process

VirtIO Memory Ballooning

VirtIO provides Memory Ballooning: the host system can reclaim memory from virtual machines (VM) by telling them to give back part of their memory to the host system. This is achieved by inflating the memory balloon inside the VM, which reduced the memory available to other tasks inside the VM. Which memory pages are given back is the decision of the guest operating system (OS): It just tells the host OS which pages it does no longer need and will no longer access. The host OS then un-maps those pages from the guests and marks them as unavailable for the guest VM. The host system can then use them for other tasks like starting even more VMs or other processes. (https://pmhahn.github.io/virtio-balloon/)

Agent-based metrics

- *swap_in, swap_out* -The number of swapped-in and swapped-out pages as reported by the guest OS since the start of the VM.
- *major_fault, minor_fault* - The number of page faults as reported by the guest OS since the start of the VM.
- *unused* - Inside the Linux kernel this actually is named MemFree.
  usable - Inside the Linux kernel this is named MemAvailable
- *available* - Inside the Linux kernel this is named MemTotal.

Metrics
Network

Libvirt metric name	Unit	Collectd metric name	Description
net.count	count		Number of network interfaces on this domain
net.N.name	string	tag in metric	Interface name
net.N.rx.bytes	bytes	collectd_virt_if_octets_rx_total{virt="name"}	Bytes received https://libvirt.org/html/libvirt-libvirt-domain.html#virDomainInterfaceStatsStruct
net.N.rx.pkts	packets	collectd_virt_if_packets_rx_total{virt="name"}	Packets received
net.N.rx.errs	packets	collectd_virt_if_errors_rx_total{virt="name"}	Receive errors
net.N.rx.drop	packets	collectd_virt_if_dropped_rx_total{virt="name"}	Receive packets dropped
net.N.tx.bytes	bytes	collectd_virt_if_octets_tx_total{virt="name"}	Bytes transmitted
net.N.tx.pkts	packets	collectd_virt_if_packets_tx_total{virt="name"}	Packets transmitted
net.N.tx.errs	packets	collectd_virt_if_errors_tx_total{virt="name"}	Transmission errors
net.N.tx.drop	packets	collectd_virt_if_dropped_tx_total{virt="name"}	Transmit packets dropped

- Network metrics are available per network interface.
- Name of network interface is the same as on host system. Metrics also can be retrieved from ifconfig <interface> with swap of rx and tx fields.

Metrics
Disk

Libvirt metric name	Unit	Collectd metric name	Description
block.count	count		Number of block devices
block.N.name	string	tag in metric	Name of the block device <num> as string. Matches the target name (vda/sda/hda) of the block device.
block.N.path	string		String describing the source of block device <num>, if it is a file or block device
block.N.rd.reqs	count	collectd_virt_disk_ops_read_total{virt="name"}	Number of read requests
block.N.rd.bytes	byte	collectd_virt_disk_octets_read_total{virt="name"}	Number of read bytes
block.N.rd.times	nanosecond	collectd_virt_disk_time_read_total{virt="name"}	Total time (ns) spent on reads
block.N.wr.reqs	count	collectd_virt_disk_ops_write_total{virt="name"}	Number of write requests
block.N.wr.bytes	byte	collectd_virt_disk_octets_write_total{virt="name"}	Number of written bytes
block.N.wr.times	nanosecond	collectd_virt_disk_time_write_total{virt="name"}	Total time (ns) spent on writes
block.N.fl.reqs	count		Total flush requests
block.N.fl.times	nanosecond		Total time (ns) spent on cache flushing
block.N.allocation			Offset of the highest written sector
block.N.capacity	byte		Logical size in bytes of the block device backing image
block.N.physical	byte		Physical size in bytes of the container of the backing image

Datasets

Hosts

Server Machine Dataset (**SMD**) Download raw datasets

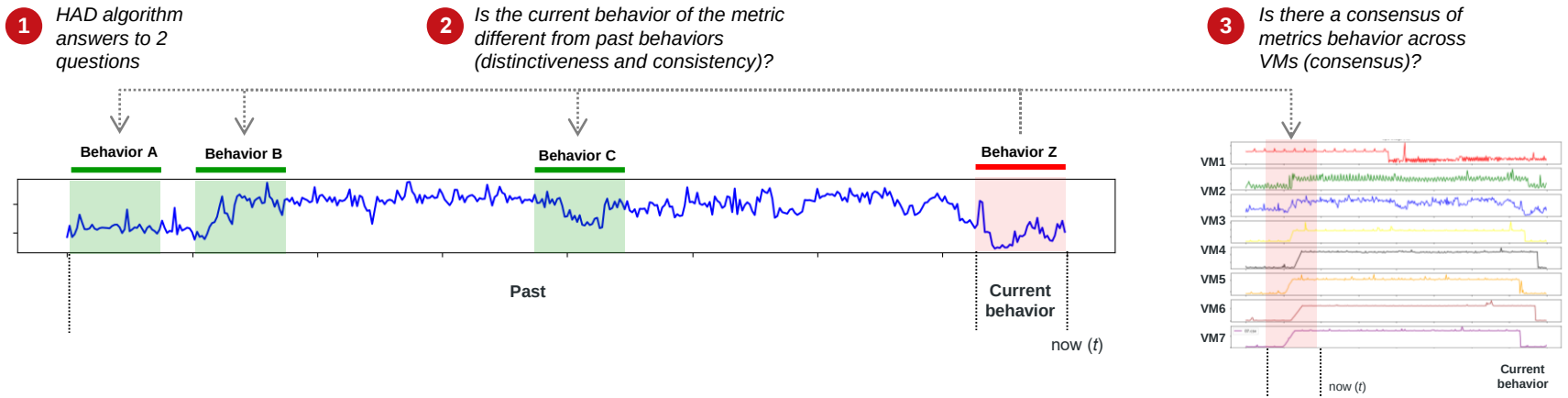
- Collected from a large Internet company containing a 5-week-long monitoring KPIs of 28 machines. The meaning for each KPI could be found here.

Algorithm Design
Attribution Theory

Reasoning

- Use historical data to learn historical behavior of metrics
  - Determine distinctiveness and consistency
  - E.g., Historical behavior = [A, A, B, C, Z, A, B]
- At time *t*, compare current behavior of VMs
  - Hypervisor is faulty if:
    - high consensus and
    - (low consistency and/or high distinctiveness)

Observation	Interpretation	Attribution	Root Cause
Does the metric behaves this way when executing other workloads?	Yes. Low distinctiveness	Internal	Workload
	No. High distinctiveness	External	**Hypervisor**
Do other metrics behave the same way when a hypervisor fails?	Yes. High consensus	External	**Hypervisor**
	No. Low consensus	Internal	Workload
Does metrics behave this way historically?	Yes. High consistency	Internal	Workload
	No. Low consistency	External	**Hypervisor**



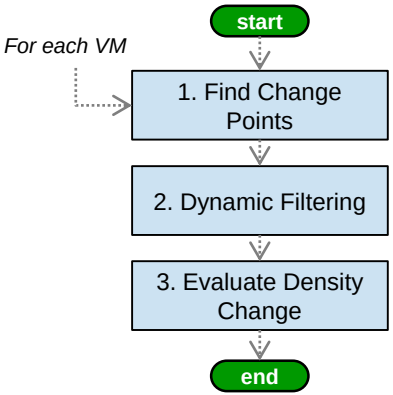
Note: Ideally, historical behavior should not include previous hypervisor failures. Once an hypervisor fails, its history needs to be deleted

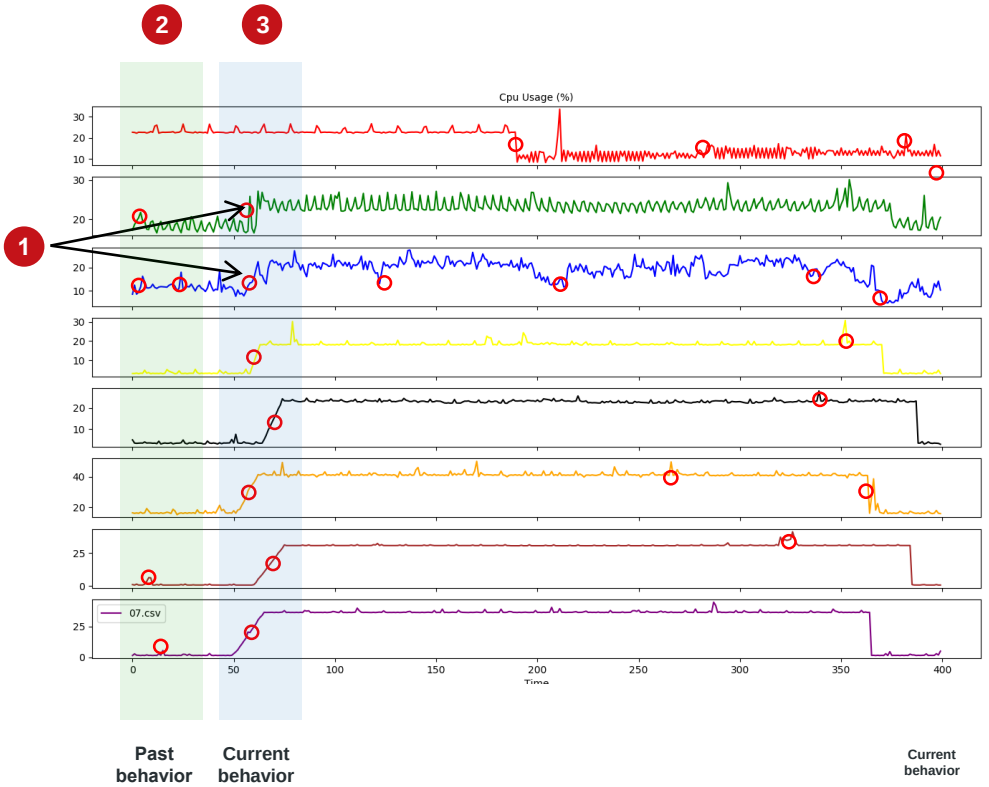
Algorithm Design
Procedure

Procedure

1. For each time series, find its change points
2. Using past behavior, dynamically filter change points
3. Find an area with a higher change density of change points across the various time series (*consensus*)

The presence of such an area is attributed to an hypervisor failure if it has not occurred in the recent past (*distinctiveness* and *consistency*)





Challenges

- Parameter selection
  - thresholds, window size (3), …
- Requires a set of online algorithms

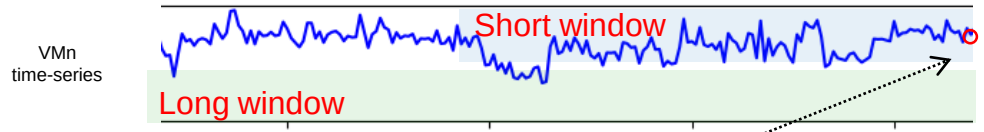
Algorithm Design
Change Point detection

Moving Z-score. Score outliers in a univariate and sequential dataset, i.e., a time series. Fits a moving average to a univariate time series and identifies points that are far from the fitted curve.

Change Point techniques
- Window-based (e.g., Z-score) O(n)
- PELT (changes in mean, variance of time series) O(n)
- bcp package (Bayesian single change point analysis of univariate time series)
- Binary segmentation O(n log n)
- Multivariate CP detection
- EWMA

Approach
- The moving Z-score for a data point x_t is simply the value of x_t standardized by subtracting the moving mean just prior to time t and dividing by the moving standard deviation just prior to t
- We use a variation of Z-score, by sampling the short window instead of using only data point x_t, this make the approach more resilient to outliers

1. Change Point detector = *z-score*(long, short)



$$Z(x_i) = \frac{x_i - \bar{x}_i}{s_i}$$

Approach
- The moving Z-score for a data point x_t is simply the value of x_t standardized by subtracting the moving mean just prior to time t and dividing by the moving standard deviation just prior to t
- We use a variation of Z-score, by sampling the short window instead of using only data point x_t, this make the approach more resilient to outliers .

moving standard deviation

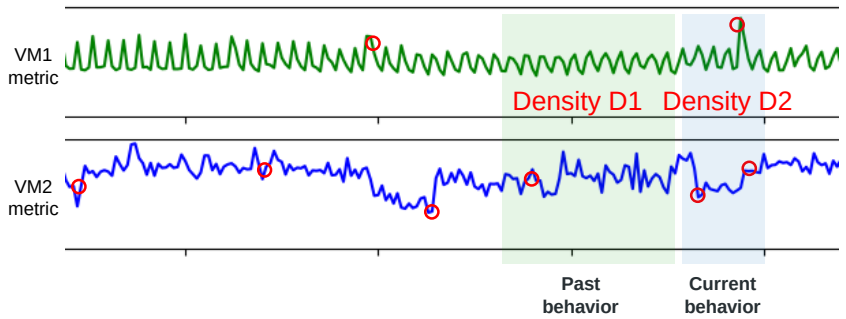
$$\bar{x}_i = \frac{1}{w}\sum_{j=i-w}^{i-1} x_j$$

moving average

Algorithm Design
Dynamic Filtering

Problem. When *c* change points are detected in a time series, is this observation historically consistent? The historical behavior of a time series needs to be considered to determine if the current behavior is different or not.

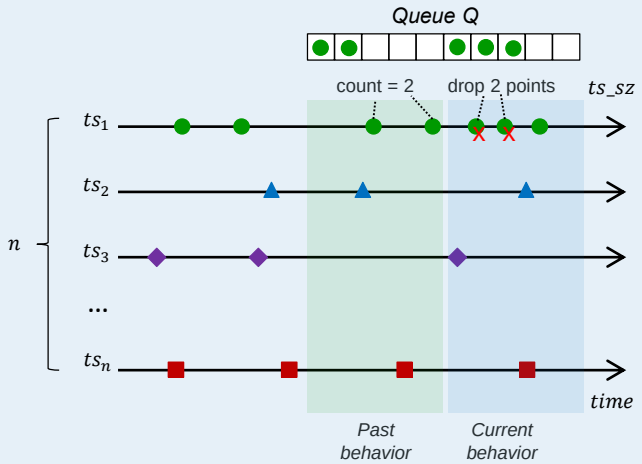
Filtering techniques
- Streaming Histograms
- Kolmogorov–Smirnov test
- Anderson-Darling (AD) test



Approach
- Use a simple form of past behavior modeling by counting the number of change points which occurred in the past during the same interval of time
- When a new change data point is detected, drop the point if it is below the count of past behavior

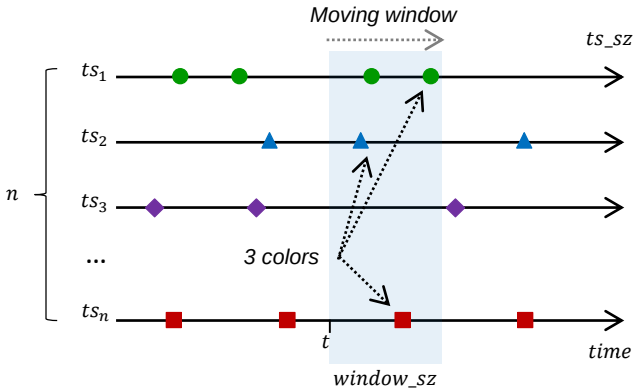
Solution. $O(n)$

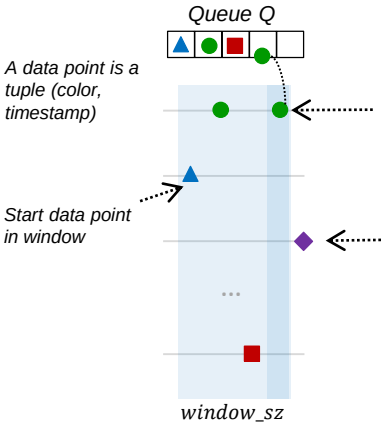
```
function dynamic_filtering(queue, dp, threshold, mid):
   if queue is full:
     count = [1 if v > threshold else 0 for v in queue]
     past_behavior = sum(count[:mid])
     curr_behavior = sum(count[mid:])
     if curr_count < past_count:
       return True
   return False
```



Algorithm Design
Correlation strength

Problem (Min Color Count Over Intervals Problem (MinColorCount)). Given n time series ts_i , of lengths ts_{sz}, find the intervals with a maximum lengths of $window\_sz$, which contain observations from c time series.





New point arrives: update(x)

- *Case 1.* distance between the timestamp of the new point and the start data point is < *windows_sz*
  - Enqueue data point *x* in Q
- *Case 2.* distance between the timestamp of the new point and the start data point is >= *windows_sz*
  - Until the first point distance is lower than *windows_sz,* dequeue Q
  - Enqueue *x*
- If queue *q* has more then *c* colors, emit queue state

Trivial solution. $O(n \times ts_{sz} \times window\_sz)$

```
function Overlap(x): (*)
  for t in [0, ts_sz]:
    color[0, n] = 0
    for w in [t, t + window_sz]:
      for i in [0, n]:
        color[a[i][w]] = i
    if |set(color[j] != 0)| > c:
      print(t, color)
```

Online solution. $O(n \times ts_{sz})$

```
function Update(x):
  while q is not empty:
    if x.timestamp - q[0].timestamp >= windows_sz:
      dequeue q
  enqueue x in q
  if |set(q[j].color)| > c:
    print(q)
```


(*) Without loss of generalization, assume that all time series have the same length

Algorithm Design
Interface and Parameters

Parameters

- Parameters are derived from the implementation of the 3 features (change point detection, filtering and density change evaluation)

Parameter List

- *window_long_sz*, *window_short_sz*
  - 2 temporal windows used to detect change points over time series
- *threshold*
  - Define the magnitude of a change point to be considered relevant due to its magnitude
- *dynamic_filtering*
  - To reduce the number of false positives, dynamic filtering can be activated
- *overlap_min_pct*
  - Percentage of time series (i.e., VMs) which need to be correlated to trigger an hypervisor anomaly
- *overlap_window_sz*
  - Windows size used to detect a correlation between time series
- *online*
  - Execute HAD as using a stream paradigm

```
class HAD(TimeseriesAlgorithm):
    """
       Hypervisor Anomaly Detection (HAD) using Change Points,
       dynamic filtering and Min Color Count Over Intervals
    """
    name = 'had'
    version = '1.0'
    multivariate_test = True

    default_config = DEFAULTS

    @typechecked
    def __init__(self,
                 window_long_sz: int = DEFAULTS['window_long_sz'],
                 window_short_sz: int = DEFAULTS['window_short_sz'],
                 threshold: float = DEFAULTS['threshold'],
                 dynamic_filtering: bool = DEFAULTS['dynamic_filtering'],
                 overlap_min_pct: float = DEFAULTS['overlap_min_pct'],
                 overlap_window_sz: float = DEFAULTS['overlap_window_sz'],
                 online: float = DEFAULTS['online']
                 ):
...
```

Metrics Collector

Download OS image

- https://ubuntu.com/download/server

Create a VM

```
virt-install --name ubuntu-1 \
--memory 2048 \
--vcpus 2 \
--disk size=8 \
--cdrom /home/jcardoso/Downloads/ubuntu-23.10-live-server-amd64.iso \
--os-variant ubuntumantic
```

Configure OS

Follow the instructions on the screen to configure your new OS

Clone VMs

```
virt-clone --original ubuntu-1 --name ubuntu-2 --auto-clone
virt-clone --original ubuntu-1 --name ubuntu-3 --auto-clone
virt-clone --original ubuntu-1 --name ubuntu-4 --auto-clone
```

Install node exporter

```
sudo apt-get update
sudo apt-get install prometheus-node-exporter -y
sudo apt-get install prometheus-libvirt-exporter -y
sudo systemctl status prometheus-node-exporter
```

libvirt metrics: curl http://localhost:9177/metrics
OS metrics: curl http://localhost:9100/metrics

Injecting Stress

stress-ng --cpu 1 --cpu-cores 0,2
In this example, --cpu 1 instructs stress-ng to create 1 worker for CPU stress, and --cpu-cores 0,2 specifies that it should stress the CPU cores 0 and 2. Adjust the number of workers and the list of CPU cores based on your requirements.

cpulimit
The cpulimit command allows you to limit the CPU usage of a process.
# Install cpulimit on Debian/Ubuntu
sudo apt-get install cpulimit

Limit CPU usage of process to 50%
cpulimit -e process_id -l 50

Workflow

1. Each VM is set to a CPU load of 50%:
stress-ng --cpu 2 --cpu-load 50 --timeout 60s

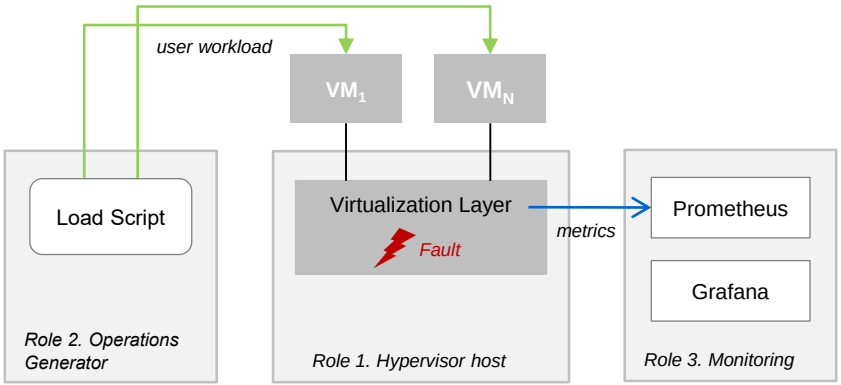
2. Limit CPU usage in qemu processes
for pid in `pgrep qemu-system-x86`; do echo $pid && sudo cpulimit -b -l 50 -p "$pid" ; done

3. Stop CPU limit
sudo killall cpulimit

```
#!/bin/bash

# Set the duration for each step (in seconds)
duration_per_step=5
n_cpu=2


# Function to run stress-ng with CPU stressor and specified load
run_stress_ng() {
    local cpu_load=$1

    if [ -z "$cpu_load" ]; then
        echo "Load not assigned."
        # Generates a random number between 0 and 99
        cpu_load=$((RANDOM % 100))
    fi
    echo "Load has a value: $cpu_load"
    stress-ng --cpu $n_cpu --cpu-load $cpu_load --timeout ${duration_per_step}s
}

echo "Simulating workload increase and decrease..."

for ((i = 1; i <= 10; i++)); do
    run_stress_ng
done

echo "Workload simulation completed."
```

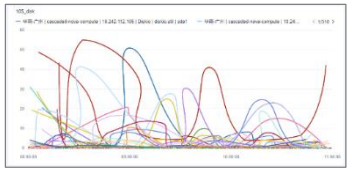
Experimental Testbed
Overview

Testbed overview

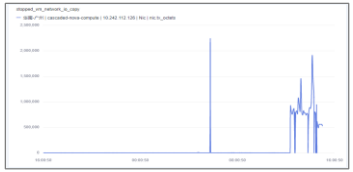


Metrics similar to available in the Cloud

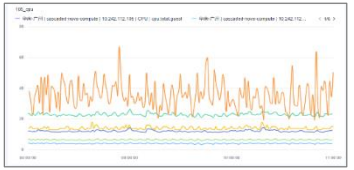
Number of VMs



Disk IO



Network IO



CPU utilization

Node roles

1. Hypervisor host (KVM with VMs running on top of it)
2. Workload Generator (emulates user traffic for applications running in VMs)
3. Monitoring (metrics processing)

Fault Injection

Faults are injected to virtualization layer of hypervisor host: to hypervisor itself, host OS, virtual volumes and virtual networking

Type of VM workloads

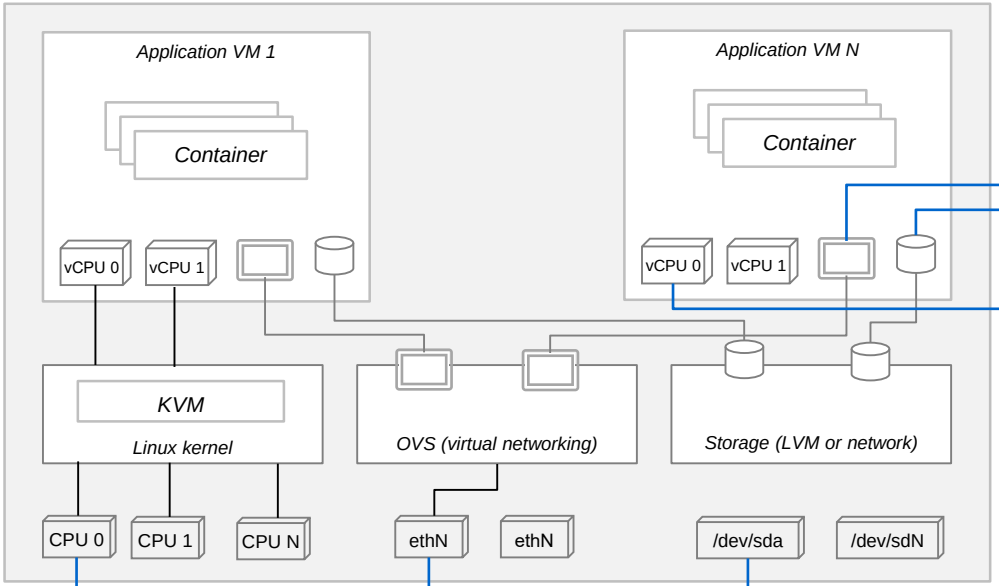
1. Online shop: front-end, back-end and database. User requests are random
2. Monitoring database with uniformly-distributed write operations
3. CPU-bound operations, short-live with high CPU consumption and low IO
4. Almost idle: low CPU and low IO

Scenarios

1. Constant workload type, constant number of VMs: one case per type, plus one mixed consisting of VMs of different types of workload.
2. Constant workload, number of VMs changes over the time, e.g. some spawned, some removed.
3. Number of VMs changes and workload changes too.

Experimental Testbed

Virtualization Platform and Metrics



Virtual Machines

Hypervisor

Physical Layer

Virtual Machine metrics

vcpu.N.time
Virtual CPU time spent by virtual CPU (https://libvirt.org/html/libvirt-libvirt-domain.html#virVcpuInfo)

net.N.rx.(bytes|pkts|errs|drop)
Received bytes, packets, errors or dropped
net.N.tx.(bytes|pkts|errs|drop)
Transmitted bytes, packets, errors or dropped

block.N.rd.(reqs|bytres|times)
Number of read requests, bytes read and time spent on reads
block.N.wr.(reqs|bytres|times)
Number of write requests, bytes written and time spent on writes

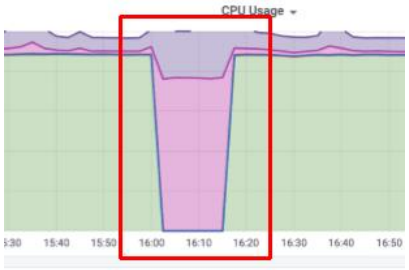
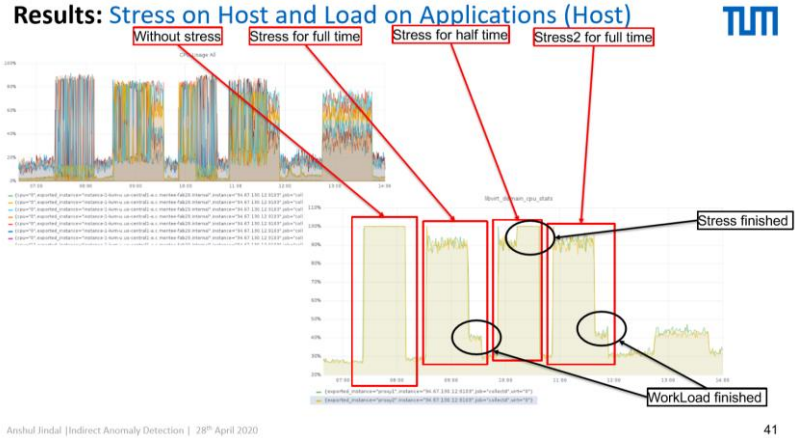
Host Metrics

cpu.(time|user|system|wait|idle)
System CPU metrics including user-space, kernel and i/o

net.N.rx.(bytes|pkts|errs|drop)
Received bytes, packets, errors or dropped
net.N.tx.(bytes|pkts|errs|drop)
Transmitted bytes, packets, errors or dropped

block.N.rd.(reqs|bytres|times)
Number of read requests, bytes read and time spent on reads
block.N.wr.(reqs|bytres|times)
Number of write requests, bytes written and time spent on writes

Experimental Testbed
Faults List

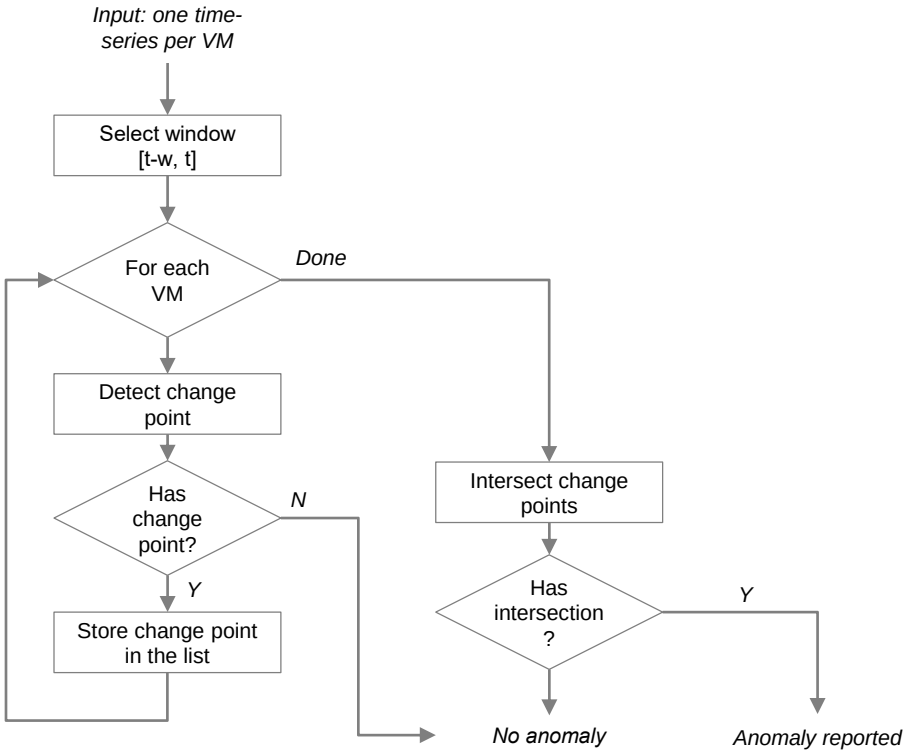
Fault	Impact on the virtualization layer	Impact on metrics (expectation)
1. **High CPU consumption by OS kernel of hypervisor host** Injection is done by using `stress-ng` tool with stressor affecting kernel space.	 Host kernel consumes a lot of CPU	

Experimental Testbed
Evaluation Framework

	Scenario	Data Generation	Algorithm	Test Module
Synthetic data	Parameters: • VM count • VM load profile (e.g. CPU consumption)	• Generate synthetic metrics and labels	• Apply algorithm	• Evaluation using labels • Calculate scores (precision, recall, F1)
Simulated data	Parameters: • VM count • VM load profile (e.g. CPU consumption) • Fault injection time • Fault strength	• Deploy VMs • Generate user workload • Inject faults and record time (used as labels) • Collect metrics from hypervisor • Download metrics		
Real cloud data		• Download the data from CloudMonitor • Manually label the data		

- *Run algorithm*
- *Evaluate*
- *Improve algorithm*

Experimental Testbed
Algorithm



Input:

- Set of time-series – one per VM

Algorithm parameters:

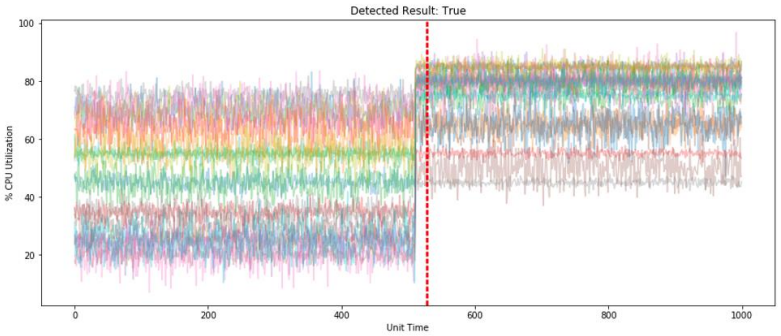
- *Window size* – number of points to take into account for change point detection
- *Change threshold* – how sensitive change point detection to feature variation

Output:

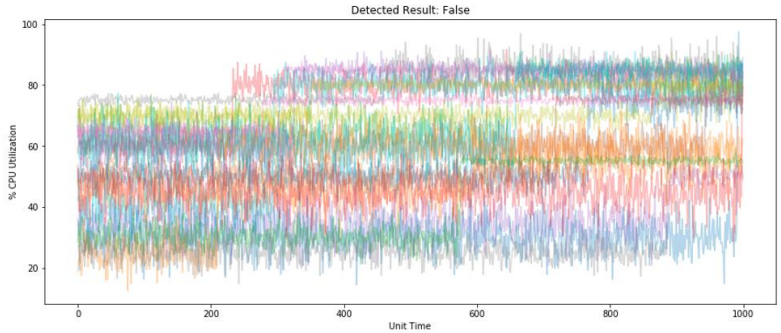
- True, if there is an anomaly in time t

Experimental Testbed
Baseline Algorithm Evaluation

Positive case



Negative Case



Results on Synthetic Data

Sr.No.	Algorithm	F1-Score	Time Taken	Acc.	Precision	Recall	TP	FP	TN	FN	TTP	TTN
1	IAD_v1(10,5)	0	27.63	0.5	0	0	0	0	5	5	5	5
2	IAD_v1(20,5)	0.57	28.33	0.7	1	0.4	2	0	5	3	5	5
3	IAD_v1(30,5)	0.75	29.18	0.8	1	0.6	3	0	5	2	5	5
4	IAD_v1(40,5)	0.75	30	0.8	1	0.6	3	0	5	2	5	5
5	IAD_v1(50,5)	0.89	30.94	0.9	1	0.8	4	0	5	1	5	5
6	IAD_v1(60,5)	0.89	32.28	0.9	1	0.8	4	0	5	1	5	5

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$

$$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

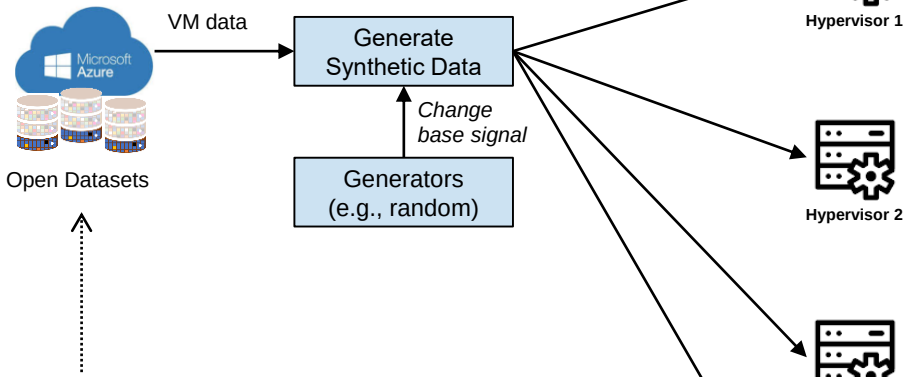
Anshul Jindal | Indirect Anomaly Detection | 19th May 2020 7

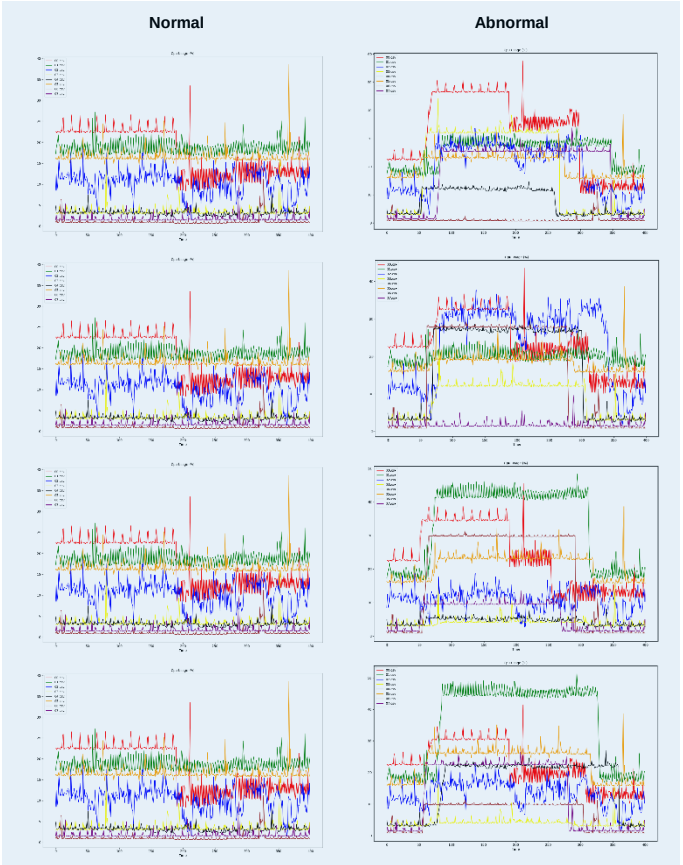
Evaluation is done with 20 time-series, change threshold 5% and window size from 10 to 60 points

Further (future) improvements

- Voting threshold < 100%
- Handle a case when change point detected not in exact time but with small variation
- Improve change point detection algorithm (more sophisticated then just mean value)

Evaluation (semi-synthetic)
Dataset Generation





Other datasets used in previous experiments

	Dataset	Time Ticks	Number of VMs	Positive Cases	Negative Cases
1	Synthetic Dataset [1]	1000	10	5	5
2	Experimental Synthetic Merged Dataset [2]	5400	2 + 8 (synthetic)	42	17
3	Microsoft Azure Dataset*[3]	5400	10	16	10
4	Alibaba Dataset*[4]	5400	10	10	10

Evaluation (semi-synthetic)
Dataset Generation

Example of a dataset generator for test case: 'random_hard'

```
def gen_random_hard():
    n_ts = 30
    return {
        'env':
            {
                'name': 'random_hard',
                'description': 'Increase 20%-50% (with delay 0-30, width 1-15) in 90% VMs',
                'n_hypervisors': 5,
                'n_vms_sets': [10, 15],
            },
        'data':
            {
                'src_data_dir': os.path.abspath('../datasets/azure/normal/'),
                'dst_data_dir': os.path.abspath('./.data/azure/random_hard/'),

                'ts_len': 575,
                'ts_anomaly_point': 50,
                'n_ts': n_ts,
                'ts_percentage_affected': .9,

                'ts_increase_width': np.random.randint(1, 15, size=n_ts, dtype=int).tolist(),
                'ts_increase_delay': np.random.randint(0, 30, size=n_ts, dtype=int).tolist(),
                'ts_increase_height_percentage': np.random.uniform(0.2, 0.5, size=n_ts).tolist(),

                'ts_stay_width': np.random.randint(200, 220, size=n_ts, dtype=int).tolist(),

                'ts_decrease_width': np.random.randint(1, 5, size=n_ts, dtype=int).tolist(),
                'ts_decrease_height_percentage': np.random.uniform(0.0, 0.0, size=n_ts).tolist(),
                'ts_decrease_delay': np.random.randint(1, 40, size=n_ts, dtype=int).tolist(),
            },
        'param_grid':
            {
                'window_long_sz': [25, 30, 40],
                'window_short_sz': [10, 20],
                'threshold': [1.6, 1.8, 2.0, 2.2],
                'dynamic_filtering': [True, False],
                'overlap_min_pct': [.7, .8, .9],
                'overlap_window_sz': [30],
                'online': [True],
            },
        'optimization':
            [
                Integer(25, 40, name='window_long_sz', dtype=int),
                Integer(10, 20, name='window_short_sz', dtype=int),
                Real(1.8, 2.1, name='threshold', dtype=float),
                Integer(0, 1, name='dynamic_filtering', dtype=int),
                Real(.6, .8, name='overlap_min_pct', dtype=float),
                Integer(40, 60, name='overlap_window_sz', dtype=int),
            ]
    }
```

Simulation environment

Data generation

Parameter grid search

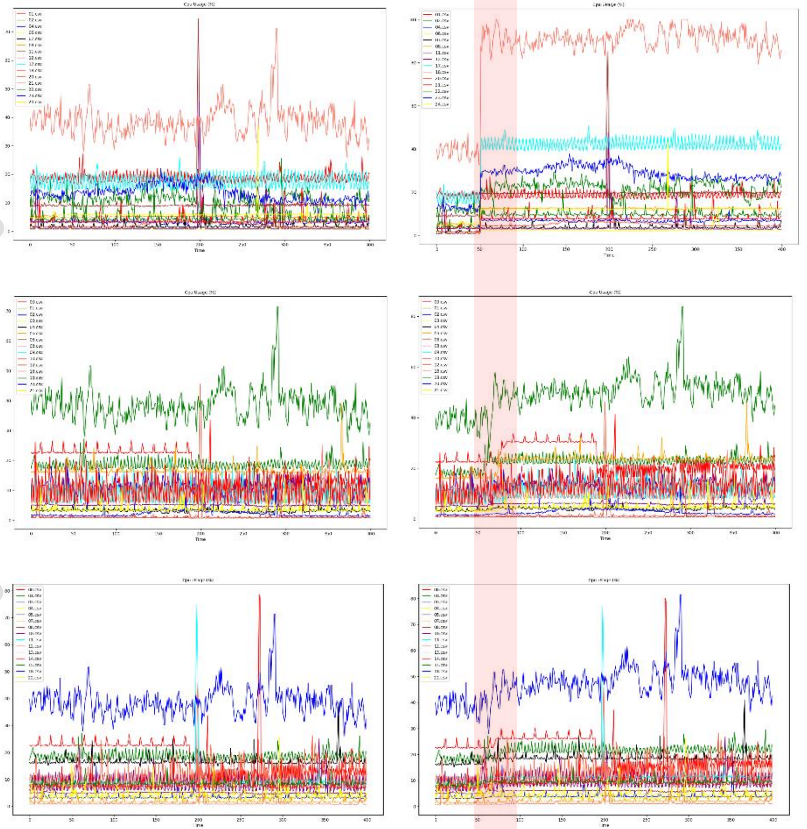
Parameter optimization

Generators

- Random easy
- Random hard
- Random difficult

Hypervisor 1

Normal | *Hypervisor anomaly* | **Abnormal**

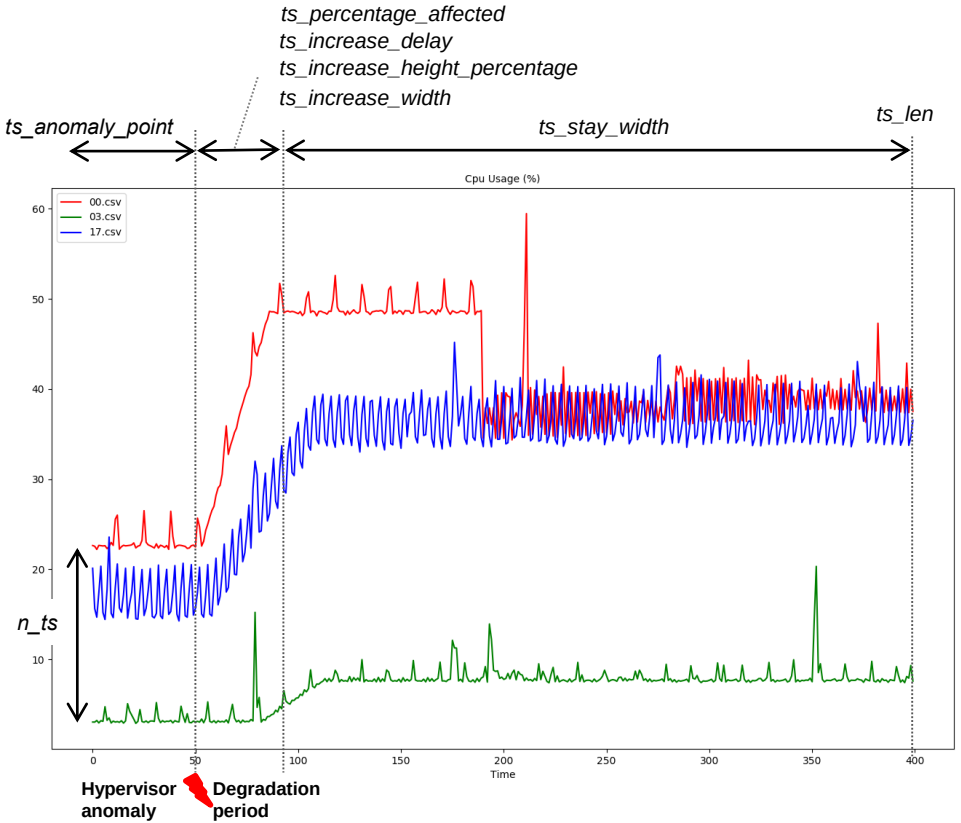


Each test case: 5 hypervisors with 30 VMs each

Evaluation (semi-synthetic)
Dataset Generation

Generators Key Parameters

- *n_ts*
  - Number of time-series (VMs) to generate
- *ts_anomaly_point*
  - timestamp when an anomaly occurs
- *ts_percentage_affected*
  - Percentage of ts affected by the anomaly
- *ts_increase_delay*
  - Number of data points after the anomaly at which VMs are affected
- *ts_increase_height_percentage*
  - Percentage of ts change after anomaly
- *ts_increase_width*
  - Width of ts change after anomaly
- *ts_stay_width*
  - Width of stay after degradation
- *ts_decrease_delay*
  - Number of data points after stay period
- *ts_decrease_height_percentage*
  - Percentage of change after stay period
- *ts_decrease_width*
  - Width of ts change after stay period

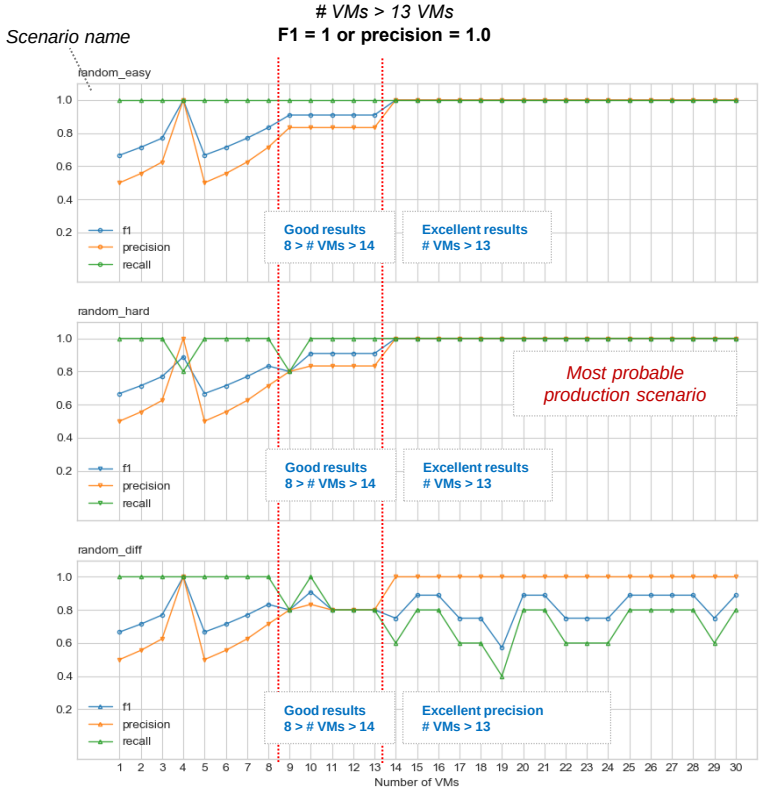


Evaluation (semi-synthetic)
F1, precision, recall

Selecting Parameters for Evaluation
- Apply Bayesian optimization on random diff(icult) scenario to identify best parameters for addressing difficult cases
- Select random set of parameters from best sorted set
- Selected parameters
  - {'window_long_sz': 30.933333333333334, 'window_short_sz': 17.8, 'threshold': 1.9886447684992719, 'overlap_min_pct': 0.7206731936373396, 'overlap_window_sz': 51.0}
- *Note: depending on the results from running HAD in production, the technique to find the pseudo optimal parameter set needs to be revised*

Procedure
- Generate 3 types of datasets for hypervisors
  - Random easy, random hard, random difficult
- Evaluation HAD on all 3 scenarios (e.g., random easy, random hard, …) by varying the number of VMs managed by hypervisor
- Calculate F1, precision and recall

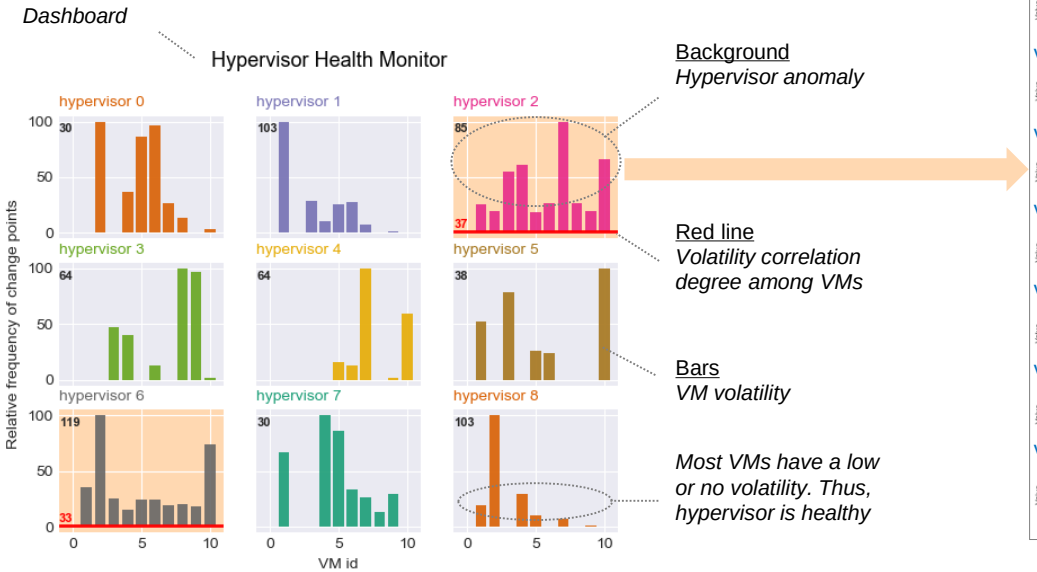
Evaluation Results
- In general, the results are excellent at # VMs > 13
- For scenario 1) and 2), precision = 1, when # VMs > 13
  - No false positives
- Recall is also good for these scenarios
  - With the current implementation, a few hypervisor failures will not be caught when the # of VMs is low (# VM < 14)



F1, precision, recall

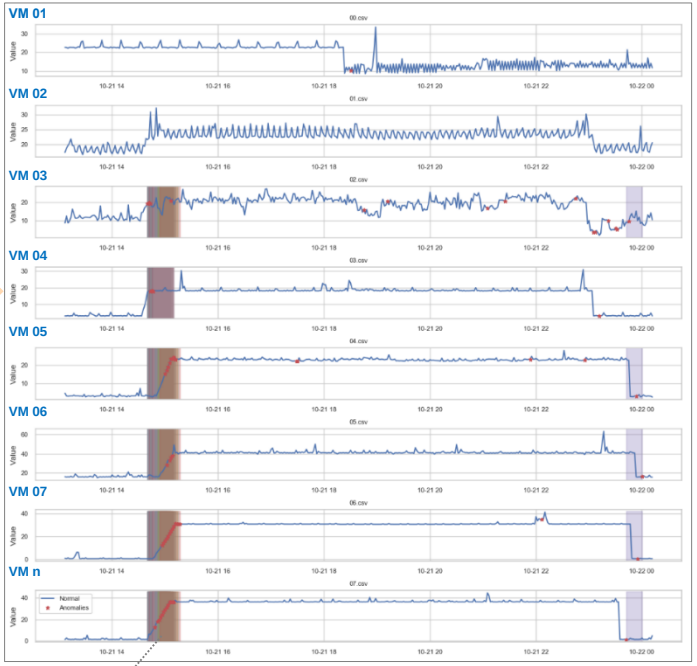
Evaluation (semi-synthetic)
User Interface

Dashboard
- Shows the heath status of hypervisors (9 hypervisors are shown)
  - Beige background indicates a problem
  - Background color indicated correlation
- Each hypervisor shows the volatility of the managed VMs
  - High bars indicate high volatility



Note: the correlation degree determines if the volatility of VMs is, or is not, related to the volatility/churn of workloads

PinPoint Visualization
- Detailed visualization on the behavior of VMs' signals
  - Change points and overlapping regions are shown



Vertical overlaps represent the correlation of behavior change as a consequence of, e.g., an hypervisor failure. The number of horizontal overlaps indicate the confidence of an anomaly.

Evaluation (semi-synthetic)
Performance: Online vs Offline

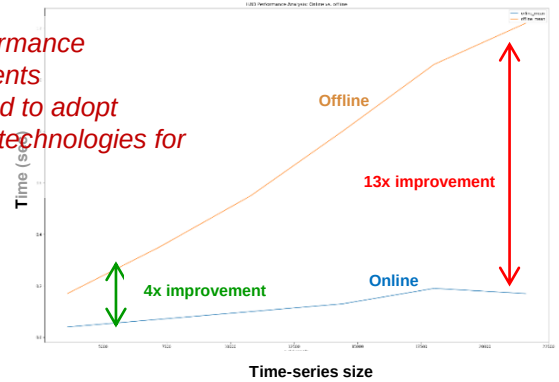
Limitations of offline HAD
- Uses 2 sliding windows: mean and std are expensive to recompute

Effect of time series size on running time
- 1 hypervisor; 1 second sampling; 25 VMs
- Monitoring duration: 1h, 2h, 3h, 4h, 5h, 6h
- Online improvements: 4x (1h) to 13x (6h)

n_datapoints	n_ts	window_long_sz	window_short_sz	improvement_pct	improvement_fold	online_mean	offline_mean
3600	25	100	25	23.96	4.17	0.04	0.17
7200	25	100	25	19.41	5.15	0.07	0.35
10800	25	100	25	17.52	5.71	0.1	0.55
14400	25	100	25	16.38	6.1	0.13	0.8
18000	25	100	25	18.25	5.48	0.19	1.06
21600	25	100	25	13.52	7.39	0.17	1.22

High performance improvements recommend to adopt streaming technologies for monitoring

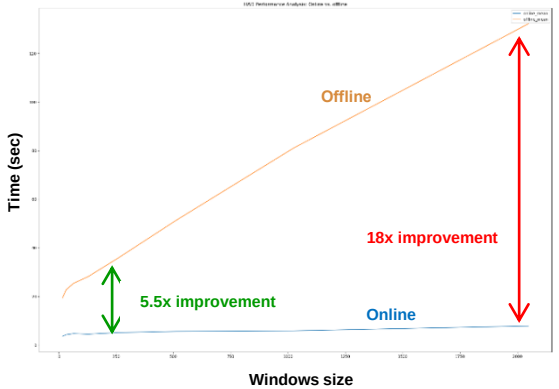
HAD Performance Analysis: Online vs. Offline (windows = (100, 25))



Effect of window size on running time
- 1 hypervisor; 1 second sampling; 25 VMs
- Sliding windows size: [16, 32, 64, 128, 256, 512, 1024, 2048], 15
- Monitoring duration: 6h
- Online improvements: 5.5x (sz: 16) to 18x (sz: 2048)

n_datapoints	n_ts	window_long_sz	window_short_sz	improvement_pct	improvement_fold	online_mean	offline_mean
80000	8	16	15	17.76	5.63	0.75	4.21
80000	8	32	15	19.32	5.18	0.87	4.53
80000	8	64	15	18.45	5.42	0.88	4.76
80000	**8**	**128**	**15**	**15.38**	**6.5**	**0.84**	**5.44**
80000	8	256	15	12.26	8.16	0.83	6.74
80000	8	512	15	9.85	10.15	0.92	9.36
80000	8	1024	15	9.57	10.45	1.35	14.11
80000	8	2048	15	5.45	18.34	1.49	27.24

HAD Performance Analysis: Online vs. Offline (8 ts * 10k data points)



Evaluation (semi-synthetic)
Conclusions

Assumptions
- CPU is the best metric to predict hypervisors' anomalies
- Hypervisor anomalies cause changes in CPU mean and std. 3 scenarios exist:
  - Increase 100%-150% (with delay 0, width 1) in 90% VMs
  - Increase 20%-50% (with delay 0-15, width 1-15) in 90% VMs
  - Increase 10%-30% (with delay 0-15, width 1-15) in 90% VMs

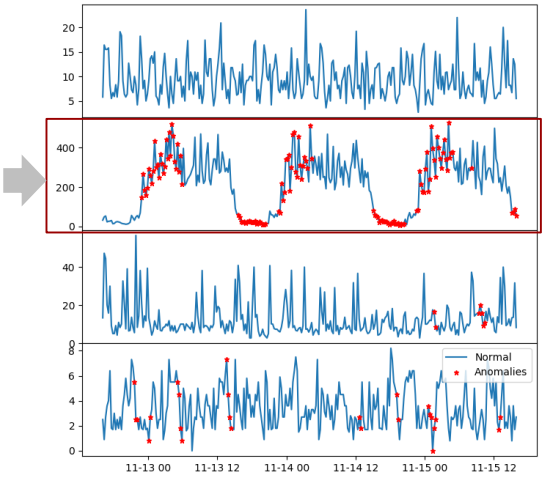
HAD algorithm
- Compare change point approaches with knee or trend identification
- Static training with holistic CPU model
- Dynamic training
- Effect of low CPU on false positives (e.g., the increase of 100% of a low metrics can be false positive dues to glitches and sudden increases of OS processing)
- Variable threshold as a function of the number of VMs running in an hypervisor

Next steps
- Evaluate HAD with production data
- Decide between using the offline or *online* implementation
- Prepare system design
- Code goes to production (to be included in CAD)

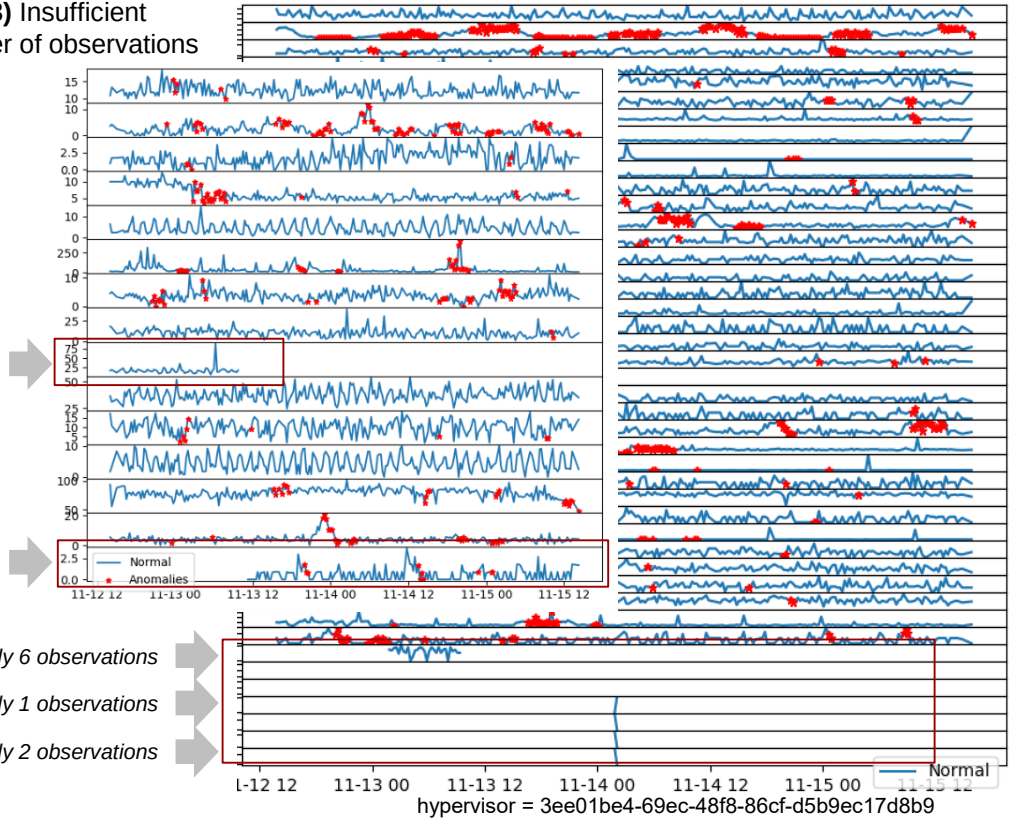
Further research
- Can hypervisor application logs also be used?
- Can kvm_stat (which monitors more than 30 hypervisor metrics) also be used?

Evaluation (CloudScope / CMC)
Insights on VM metrics

Case 1) Periodic
Case 2) Insufficient number of VMs



Case 3) Insufficient
Number of observations



Evaluation
Baseline

Sr No.	Algorithm	Input	Parameters
1	IAD	timeticks x num_vms	Window Size, Threshold, Percentage VMs Anomalous
2	ECP[5]	timeticks x num_vms	# of change points, Minimum number of observations between change points
3	BNB[6]	timeticks x num_vms	Window Size, number of trees, threshold for change points
4	BNBOnline[7]	timeticks x num_vms	Window Size, number of trees, threshold for change points
5	Isolation Forest[8]	timeticks x num_vms	contamination factor, requires training
6	Isolation Forest with Features	timeticks x num_features	contamination factor, requires training

Sr No.	Algorithm	Synthetic	Experimental Synthetic Merged	Azure	Alibaba
1	IAD	0.96	0.86	0.96	0.57
2	ECP[5]	0.67		0.76	
3	BNB[6]	0.62	0.90	0.8	0.33
4	BNBOnline[7]	0.87	0.81	0.86	0.4
5	Isolation Forest[8]	0.76	0.83	0.76	0.66
6	Isolation Forest with Features	0.76	0.83	0.76	0.66

Conclusions
Challenges

Improvements

- How to handle seasonality at low cost?
- How to handle noise?
- Update Dashboard to handle between 1 and 9 hypervisors with 8-20 VMs

Related Work

https://cran.r-project.org/web/packages/ecp/vignettes/ecp.pdf

https://bhooi.github.io/papers/bnb_sdm19.pdf

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.