



Look Who's Talking

Discovering Dependencies between Virtual Machines Using CPU Utilization

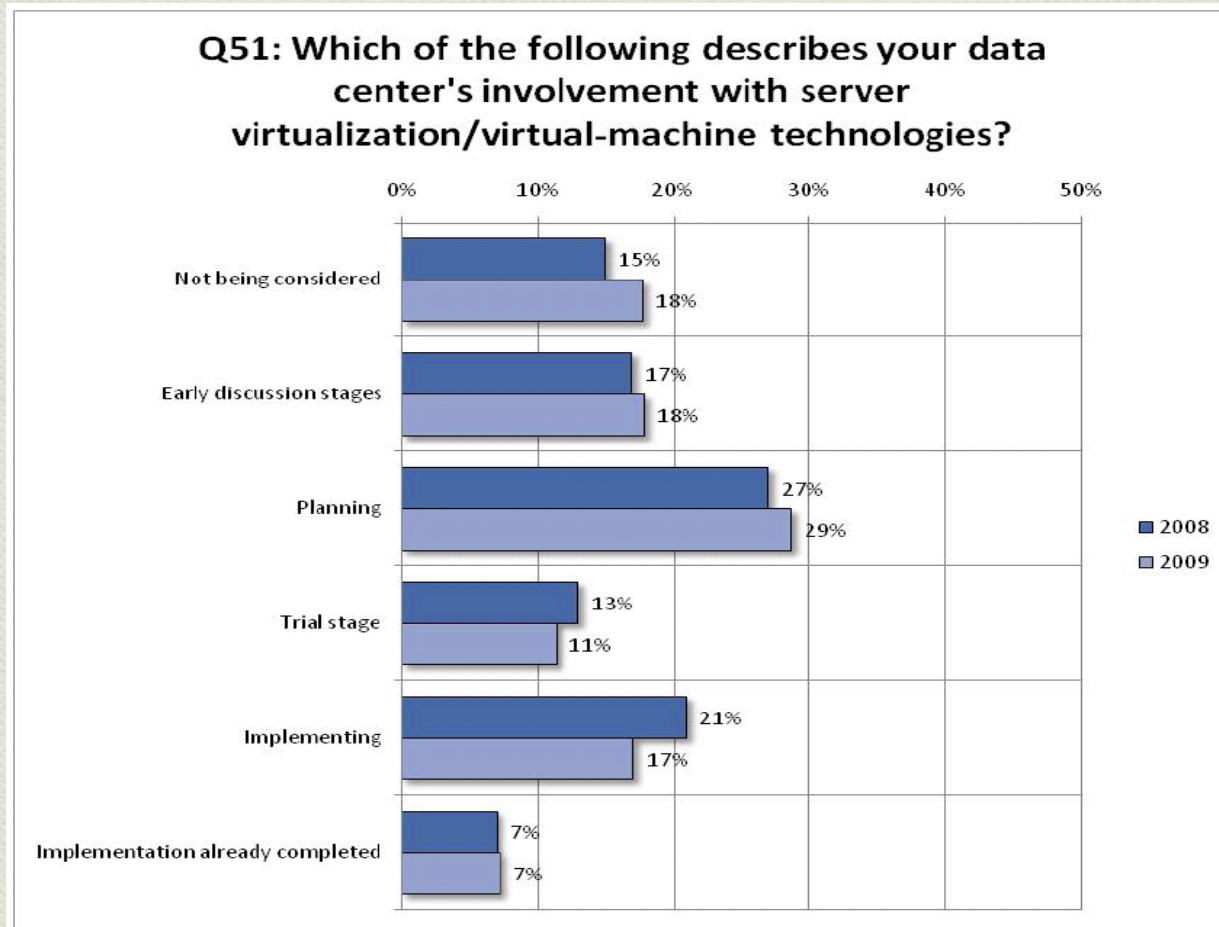
Renuka Apte, Liting Hu, Karsten Schwan, Arpan Ghosh

Georgia Institute of Technology

Talk by **Renuka Apte** *

*Currently at NVIDIA corporation

State of Virtualization Adoption



* Source: Symantec State of the Data Center Survey 2010

Challenges to Virtualization Adoption

“Manageability is top challenge in adopting virtualization”

“**Troubleshooting in the Dark:** 27 % identified a lack of visibility and tools as the largest troubleshooting challenge in virtual environments”

- SNW Virtualization Summit 09

“36% said they lacked the appropriate tools to monitor their virtual servers and desks, citing this as the greatest problem with virtualization”

- Survey of Interop 2009 participants

- Survey of Interop 2010 participants

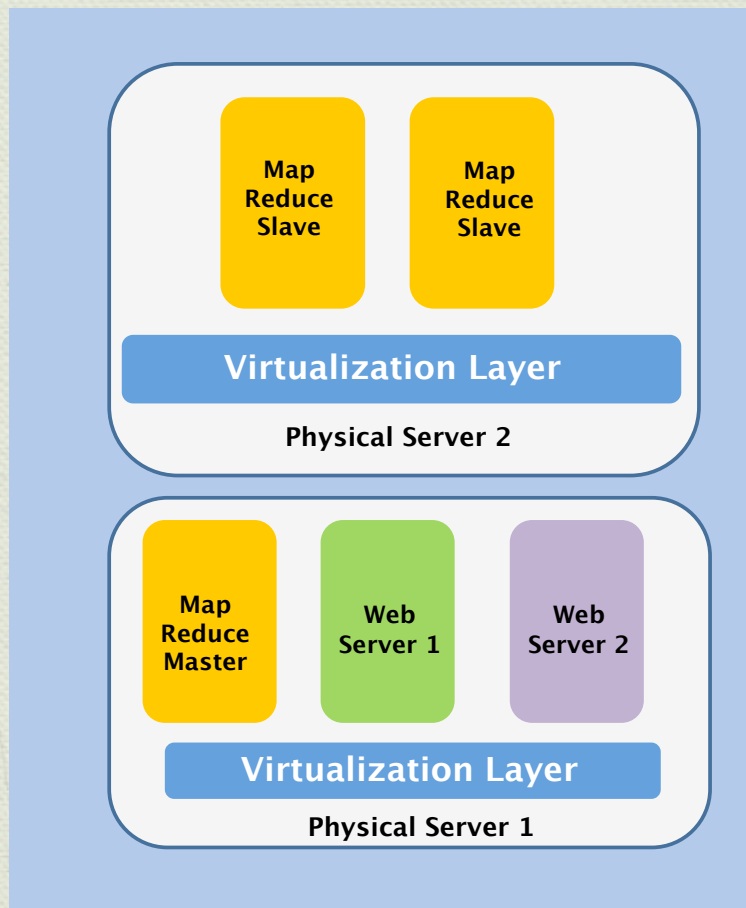
“53.9% indicated ‘VM sprawl and flexible deployment capabilities leading to unmonitored/invisible machines’ as a security concern related to virtualization”

- PRISM Microsystems State of Virtualization Security Survey

The Butterfly Effect In The Virtualized Cloud

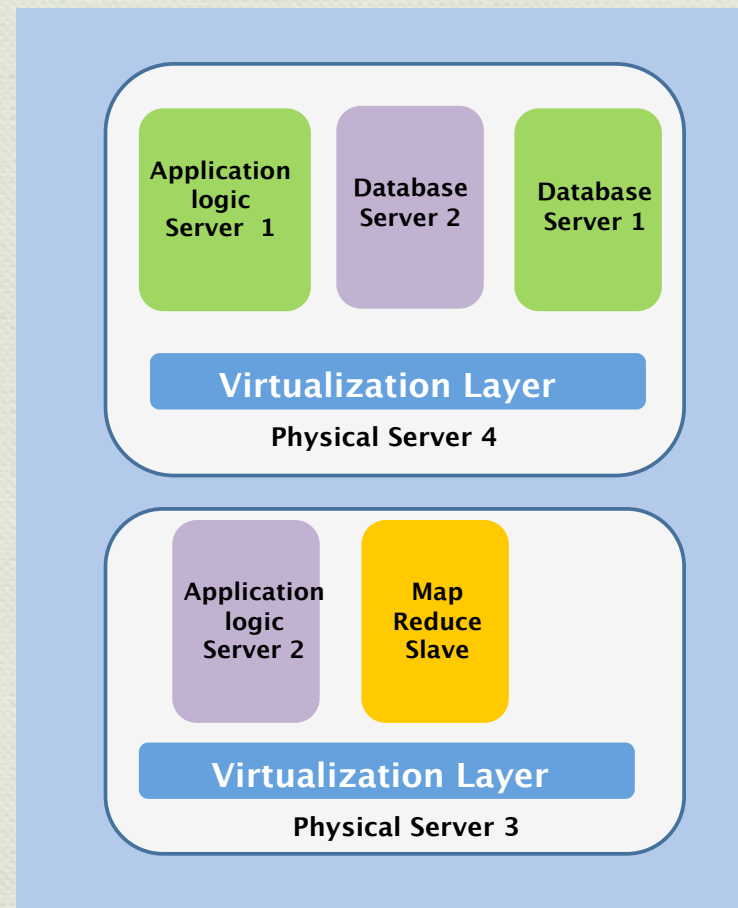
- Small variations in a complex, dynamic system...larger and more complex variations over the long term
- Problem compounded by:
 - multi-tier application infrastructure
 - VM/application inter-dependencies
 - distributed architectures
 - Dynamic creation and migration of VMs (VM Sprawl)
 - Lack of visibility into VM's workload
- VM migration, aggressive DRS and automated DR can trigger unforeseen consequences if done without realizing the 'big picture'

Typical Virtual Cloud



Rack 1

....



Rack 'n'

Dependencies Between VMs

- VM ensembles spread across multiple machines
- **Dependence relationships:** 'uses' relations in which two VMs communicate because one VM offers a service used by another

Knowledge of VM Interdependencies Helps

- Better VM placement and migration decisions
- Better resource allocation
- Better disaster recovery automation
- Better troubleshooting
 - Identify cause of failures
 - Identifying anomalies in the system

How Does LWT Achieve This?

1. Monitor

- Sample 'per VM' CPU utilization (xentop)

3. Model

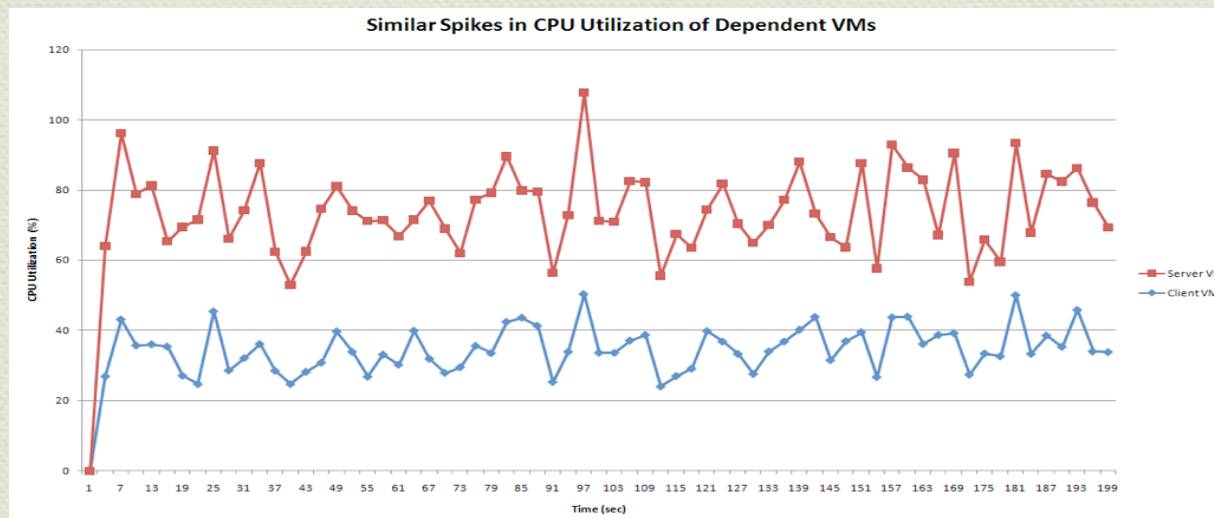
- Estimate an Auto- Regressive model for CPU utilization of each VM

4. Cluster

- K-means clusters similar AR models of interdependent VMs together

Intuition

- In a multi-tier application, VMs have request-response interactions
- The server's workload is determined by the clients workload
 - Heavier the workload of the client, the more requests it makes
 - Prominent spike in the server's CPU usage at the same time when there is a spike in the client's CPU usage



Monitoring

- CPU utilization sampled per VM using xentop
- Sampling Period
 - Too small : increases computation Too large : Might miss relevant spikes
 - Optimal period chosen as 1 sec
- Sample size
 - Increases with increasing # of VMs
 - 300 seconds: Dependency calculation can occur every ~ 5 minutes
- Perturbation
 - Dynamically change resources (CPU cycles) available to VM
 - Performance hit is reflected in dependent VMs, adds more time dependent spikes

Modeling

- Auto Regressive modeling summarizes time series CPU usage of each VM
 - Captures how one spike is influenced by previous CPU spikes
- AR model is a weighted sum of p previous values of time series dataset

$$X_t = c + \sum_p \phi_i \times X_{t-i} + \epsilon_t$$

- X_t is the CPU utilization value at time t
- ϕ are model parameters
- p is order of the model
- ϵ is white noise

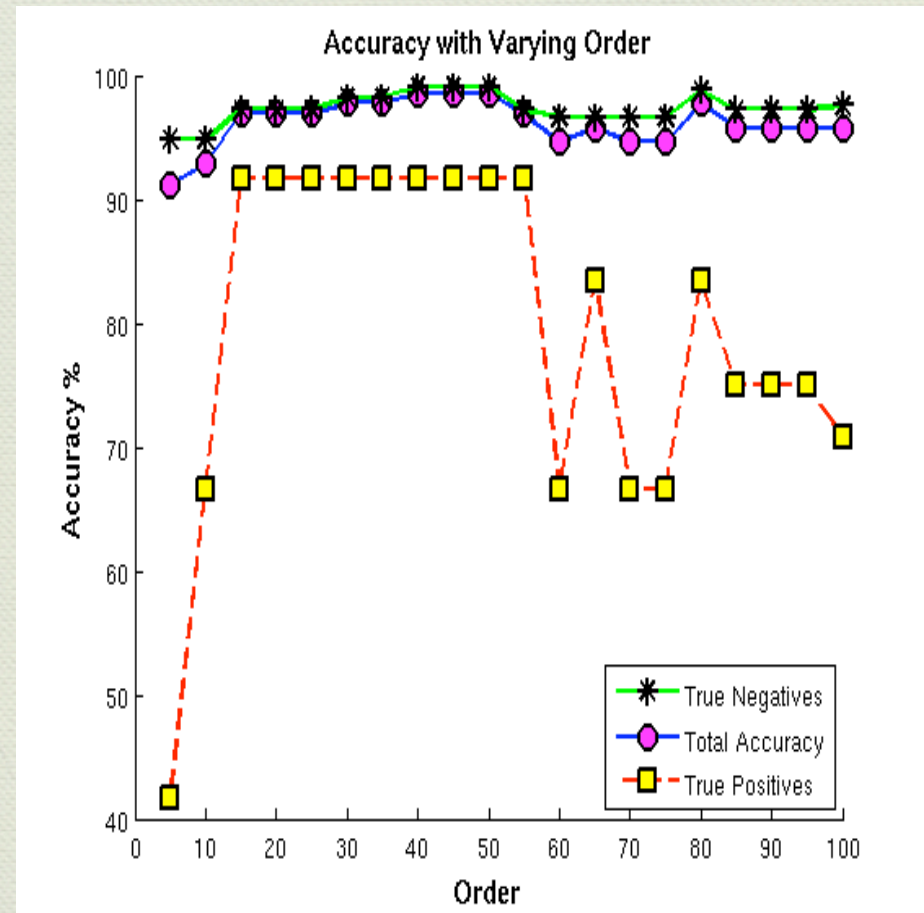
Similarity In Dependent AR Models



Coefficients of the AR models of 2 interdependent VMs

Selecting Order Of AR Model

- p will increase as system becomes more complex
 - Very large p results in over-fitting
 - 40-50 yields best accuracy for current setup

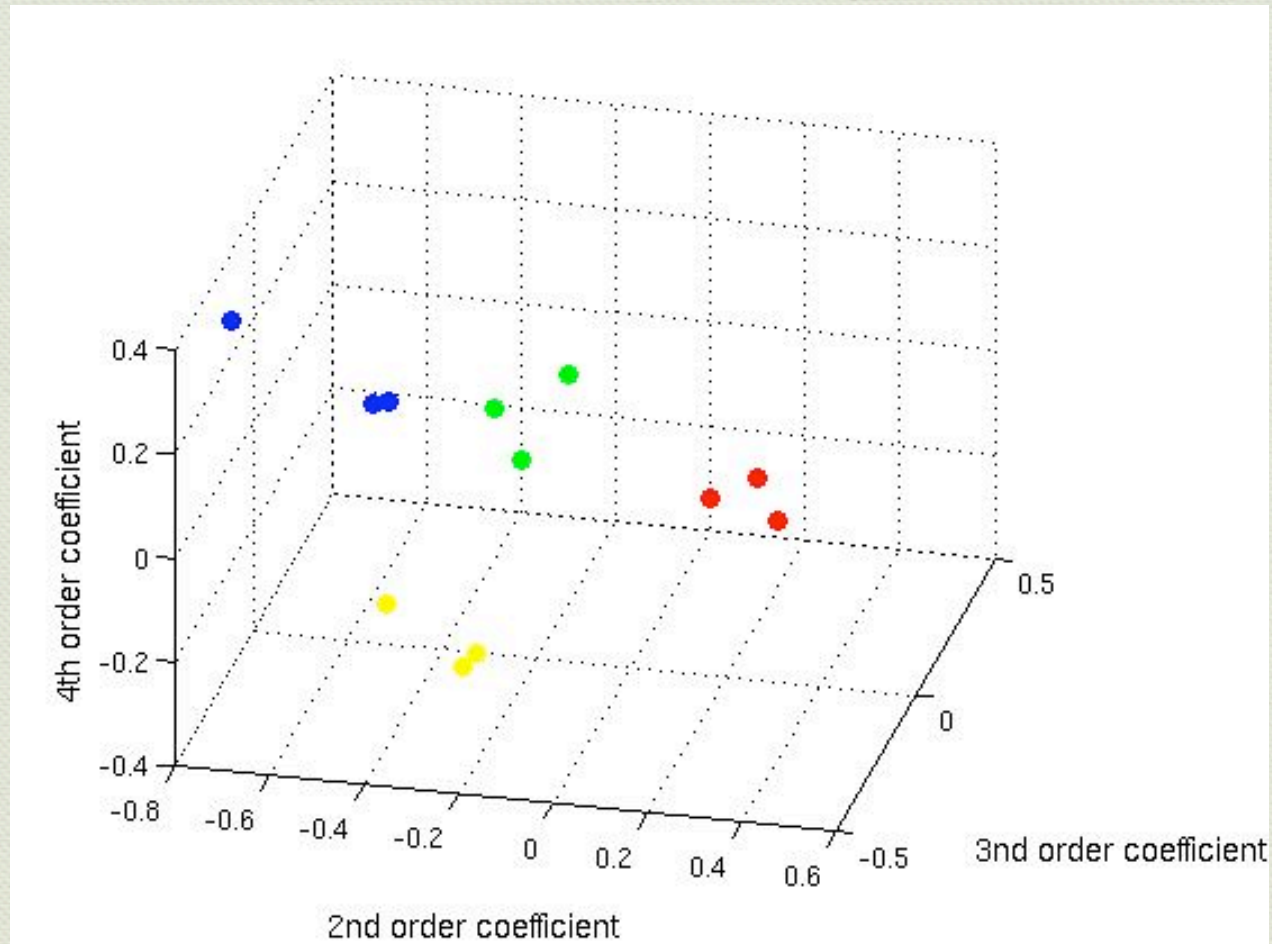


Clustering

- VMs clustered based on Euclidean distance between their AR models
 - Similar spikes at time t imply similar coefficient of X_t in AR model
 - These AR models will be closer and form cluster

- K-means divides data into K clusters
 - Iteratively selects K centroids for data
 - K is provided manually

Visualization of AR models in Space



Experimental Setup

- 31 VMs spread over 5 physical servers
 - Xen 3.1.2 virtual machine monitor
 - 512 MB RAM/ VM
- Applications/ Workloads
 - **RUBiS** : eBay like benchmark
 - Selling, browsing, bidding implemented as 3 tier application
 - An instance uses 4VMs – Apache, Tomcat, MySQL and RUBiS client
 - **Hadoop MapReduce Framework**
 - An instance uses 3 VMs – 1 master and 3 slave nodes
 - **Iperf** : Network testing tool
 - An instance uses 2 VMs – sender and receiver

Results

- Dependencies identified with overall accuracy of 97.15%
 - 91.67% true positives
 - 99.08% true negatives

Workloads		True Positives	True Negatives	False Positives	False Negatives
RUBiS	No Perturb	12	54	0	0
	Perturb	12	54	0	0
Hadoop	No Perturb	6	21	6	3
	Perturb	9	27	0	0
All	No Perturb	22	315	12	2
	Perturb	22	324	3	2

The 'All' workload consists of 3 Hadoop, 4 RUBiS and 2 Iperf instances. Total of 31 VMs

Why it works ?

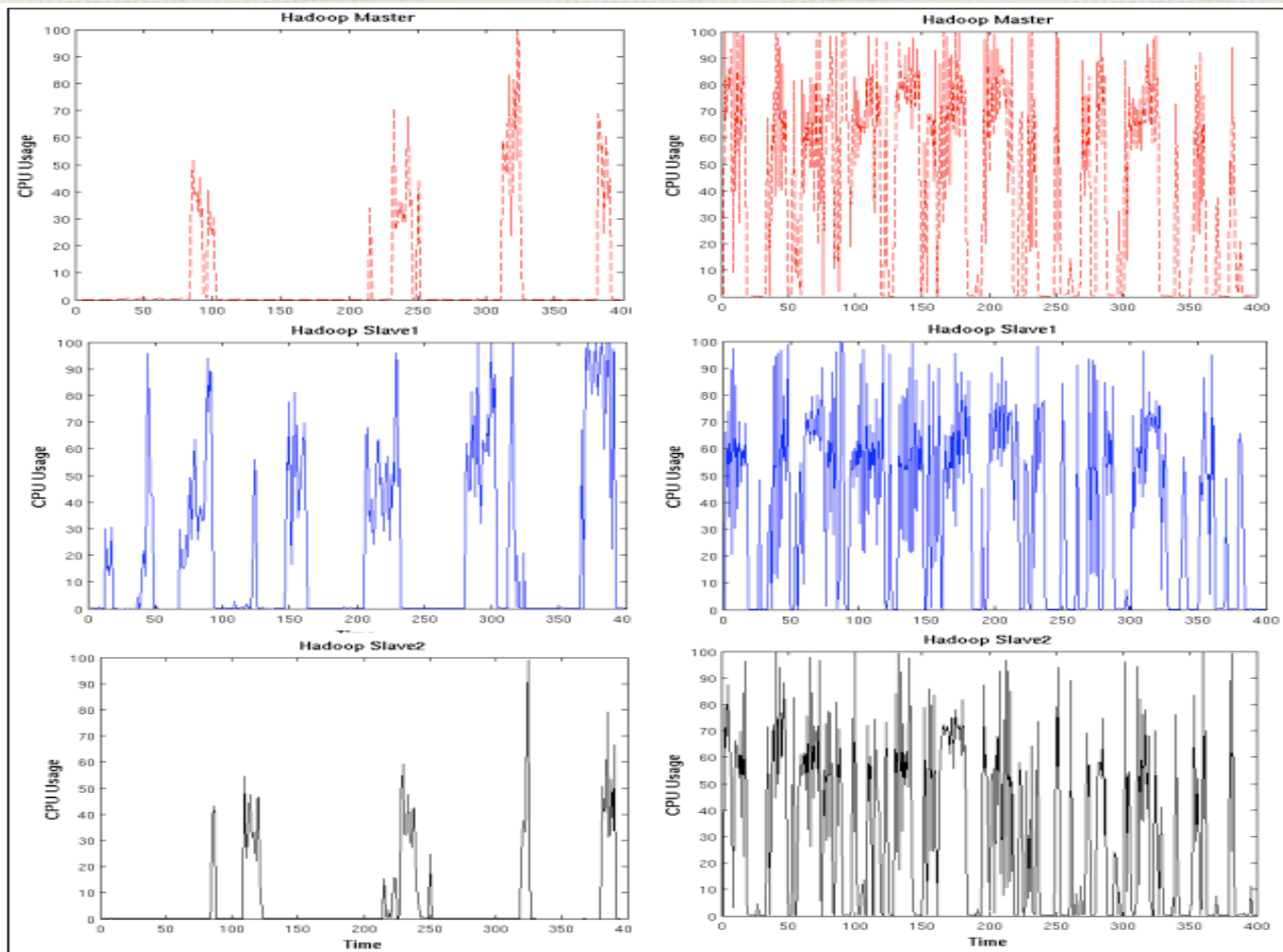
- RUBiS

- Identified dependencies with 100% accuracy
- Lot of request-response interaction between the VMs
- Follows typical 'n-tier' application model used in DCs today

- Hadoop

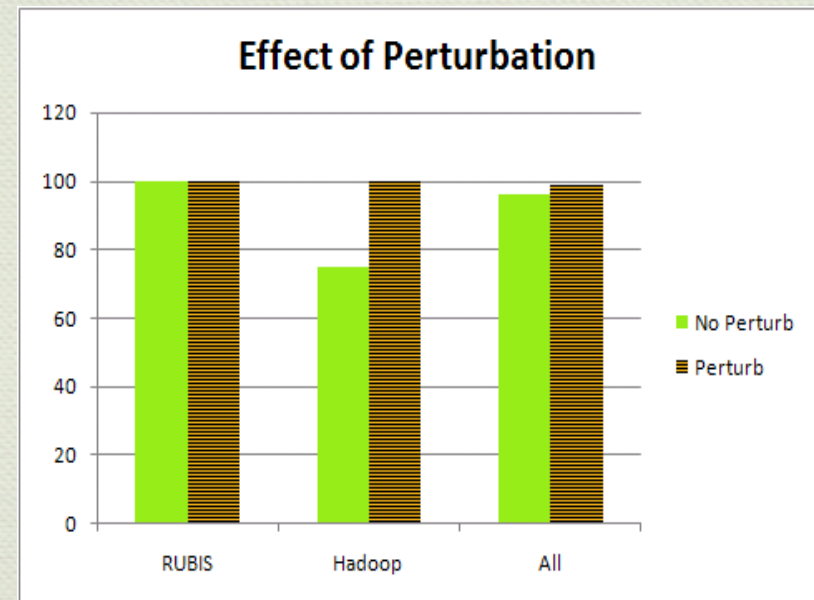
- Results more non-intuitive
- 1 master, all slaves
- Mappers and reducers communicate intermediate results via files
- Communicate to find location of input/output

Hadoop CPU usage of Dependent VMs



Perturbation



- Changed CPU cycles ('credits') available to some VMs while sampling
 - Affected performance of dependent VMs
 - Added spikes to CPU utilization of dependent VMs
- RUBiS identified 100% without perturbation
- Significant accuracy increase for Hadoop



Scalability & Time Complexity

- Time complexity depends on
 - # of VMs (N)
 - Order of AR model (p)
 - Sample size
- Finding AR models is linear in N
 - Calculated at each host and sent to central machine for clustering
- K-means complexity is $\Omega(N)$
 - Clustered a fictional dataset of 1200 VMs and $p = 100$ in 1.5 mins
 - LWT Can easily scale for a cloud DC

Conclusions

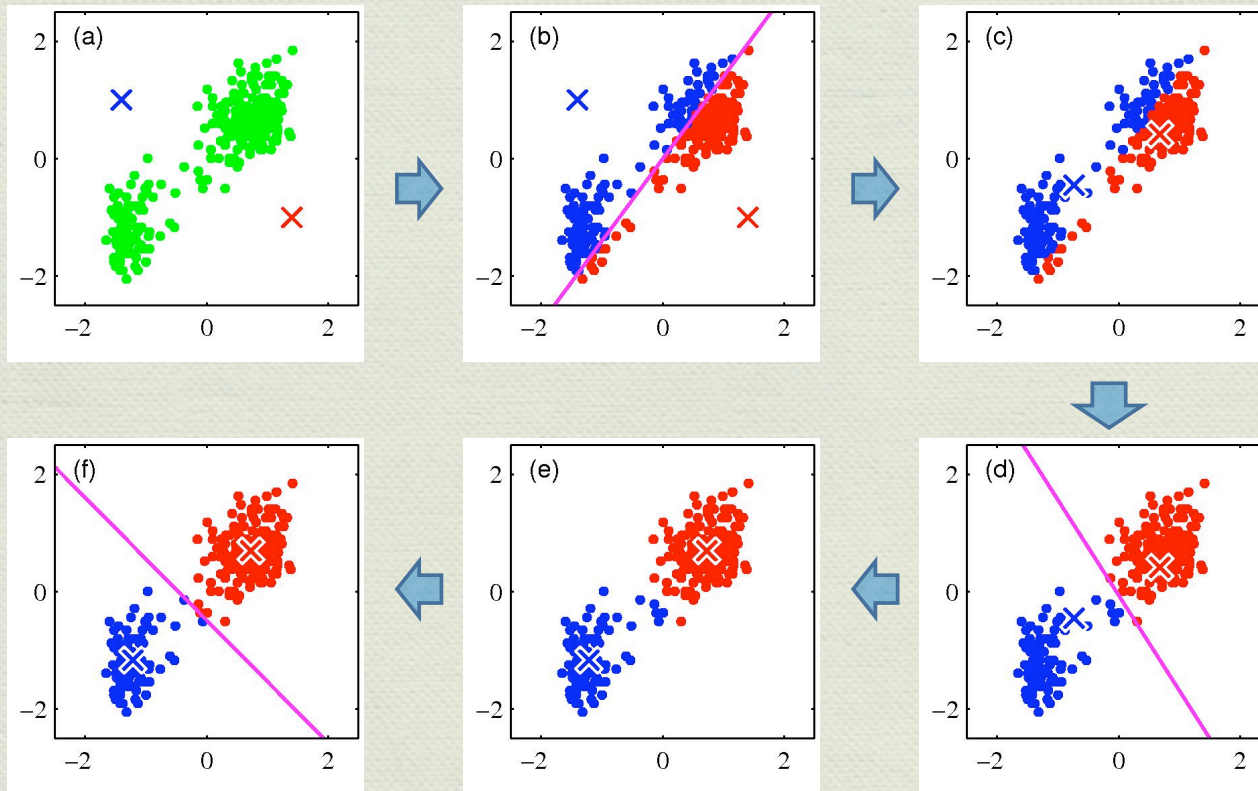
- LWT identifies inter-VM dependencies by considering only CPU usage
- LWT is non-intrusive, real-time, scalable and application agnostic
- Monitor  Model  Cluster
- 97.15% average overall accuracy

Future Work

- Deploy on large scale DC
- Add more metrics to identify dependencies accurately
- Applications where many VMs depend on 1 VM for service
- Automate calculation of sample size, AR model order, K
- Handle conditions where the initial assumption breaks

Questions?

How K-means Works?



$K = 2$. K centroids selected in each iteration

Recent Virtualized DC issues

- About 165,000 Web sites knocked offline by NaviSite outage
 - Virtual migration of servers without considering dependencies
 - Interdependent server brought up in wrong order

- Amazon EC2 hit by botnet
 - Instance compromised by Zeus botnet controller

Selection of Sampling Period

	Apache Webserver	Tomcat Server	MySQL Server	RUBiS Client	Iperf Server	Iperf Client	Nbench
Apache Webserver	1.0000	0.9823	0.9824	0.9612	0.8297	0.8373	0.8681
Tomcat Server	0.9823	1.0000	0.9941	0.9769	0.7833	0.7947	0.8642
MySQL Server	0.9824	0.9941	1.0000	0.9783	0.8046	0.8149	0.8585
RUBiS Client	0.9612	0.9769	0.9783	1.0000	0.8146	0.8247	0.8479
Iperf Server	0.8297	0.7833	0.8046	0.8146	1.0000	0.9996	0.5537
Iperf Client	0.8373	0.7947	0.8149	0.8247	0.9996	1.0000	0.5589
Nbench	0.8681	0.8642	0.8585	0.8479	0.5537	0.5589	1.0000

Correlation matrix for sampling period = 3 sec, VMs above cutoff = 0.9 are dependent

Optimal period of 1 sec determined using such matrices

Google's App Engine Datastore Failure

- Repository for determining entity location became overloaded causing read/write requests to time out
- App Engine waits 30 seconds to complete a Datastore request causing waiting requests to pile up
- Requests start to fail, regardless of whether or not they used the Datastore
- Primary and secondary Datastore out of sync
- Unapplied writes also affected the billing state of a approximately 0.3% of App Engine applications

Properties of a Dependence Discovery System

- **Near real-time**
 - DC applications and infrastructure are very dynamic
- **Non intrusive**
 - Minimal modifications to applications, OS & hypervisor
- **Lightweight**
 - Should not rob CPU/memory from VMs
- **Application & Guest OS independent**
 - Requires no knowledge of what the VM is running
- **Scalable**
- **Automated**
 - Minimal or no pre-config by admin