Look Who's Talking: Discovering Dependencies between Virtual Machines Using CPU Utilization

Renuka Apte, Liting Hu, Karsten Schwan, Arpan Ghosh

Center for Experimental Research in Computer Systems, Georgia Institute of Technology
{renuka.apte, foxting, arpan_ghosh}@gatech.edu, schwan@cc.gatech.edu

Abstract

A common problem experienced in datacenters and utility clouds is the lack of knowledge about the mappings of the services being offered to or run by external users to the sets of virtual machines (VMs) realizing them. This makes it difficult to manage VM ensembles to attain provider goals like minimizing the resources consumed by certain services or reducing the power drawn by datacenter machines. This paper presents the 'Look Who's Talking' (LWT) set of methods and framework for identifying inter-VM dependencies. LWT does not require services to be modified, or middleware or operating systems to be instrumented, but instead, operates in management VMs with privileged access to hypervisor-level information about current machine use. The current implementation of LWT has been integrated into the Xen hypervisor running across a small-scale prototype datacenter, for which experimental measurements show that it can effectively identify dependencies between VMs with an average of 97.15% overall accuracy rate, with zero knowledge of or modifications to applications or workloads and with minimal effect on system performance.

1 Introduction

"Such minor changes, such huge consequences". This famous tagline of 'The Butterfly Effect' aptly captures the dilemma most administrators of enterprise datacenters must face, due to the complexities that are inherent to these systems.

Virtualization in the cloud is becoming increasingly popular, due to advantages that include server consolidation, workload balancing, high availability, multi-tenancy and fault isolation. Recently, Cisco, EMC and VMware announced a coalition to offer organizations fully integrated, infrastructure packages that combine virtualization, networking, computing, storage, security, and management technologies with end-to-end vendor accountability [6]. Such solutions allow customers to provision new virtual cloud data centers or move existing ones, running workloads consisting of a large number of VMs, within minutes.

Typically, the applications running in such cloud data centers are composed of a number of cooperating components, running across multiple VMs. Figure 1 shows multi-tier applications deployed across three physical machines. For example, there is a multi-tier application, with an HTTP server front-end, which uses the services of an application server and a database server back-end. These VMs, typically termed *VM ensembles*, may be spread across a multitude of host machines.

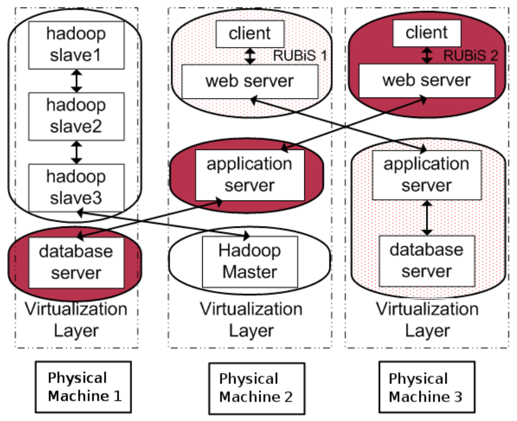
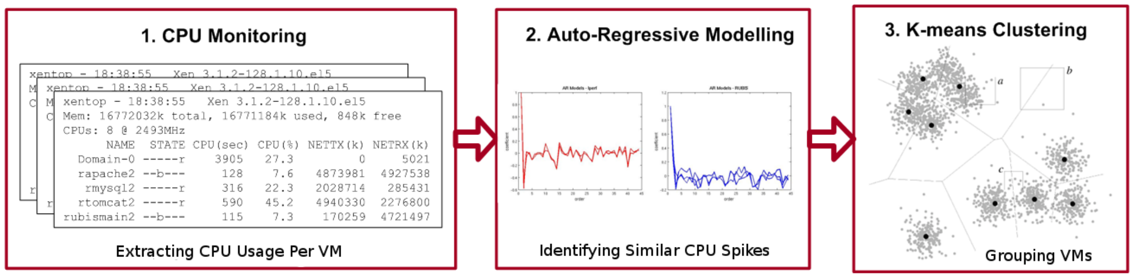


Figure 1: Multi-tier Applications Showing VM Dependencies

There are multiple *dependence* relationships between


1. CPU Monitoring
Extracting CPU Usage Per VM
2. Auto-Regressive Modelling
Identifying Similar CPU Spikes
3. K-means Clustering
Grouping VMs

```
xentop - 18:38:55   Xen 3.1.2-128.1.10.el5
Mem: 16772032k total, 16771184k used, 848k free
CPUs: 8 @ 2493MHz
       NAME  STATE CPU(sec) CPU(%) NETTX(k) NETRX(k)
   Domain-0 -----r     3905   27.3        0     5021
   rapache2 --b---      128    7.6  4873981  4927538
    rmysql2 -----r      316   22.3  2028714   285431
   rtomcat2 -----r      590   45.2  4940330  2276800
rubismain2 --b---      115    7.3   170259  4721497
```

Figure 2: LWT System Diagram

VMs within a VM ensemble, most notably those defined by 'uses' relations in which two VMs communicate because one VM offers a service used by another. For an administrator to successfully manage a datacenter – allocate resources optimally, balance & migrate workloads, meet service level agreements, detect and mitigate critical faults [9] – it is important to be able to identify the dependencies that exist between the VMs. This is because, by knowing which VMs are dependent, an administrator can make more sophisticated decisions including, but not limited to the following list:

- Whether to migrate a VM to another physical machine: if two VMs are dependent on one another, it might be preferable to keep them on the same physical machine, even if another one, physically further distant, is free.
- Whether relinquishing resources from some VM may affect another dependent service: allocation and management of resources can be made more intelligent. It might seem like a VM is over-provisioned, but the services of this VM may be used by multiple dependent VMs.
- Identifying causes of faults: based on the knowledge of dependencies, the management system can better determine cause-effect relations, and determine how a fault may manifest itself.

In this paper, we introduce the 'Look Who's Talking' (LWT) set of methods and framework for identifying inter-VM dependencies. LWT accomplishes this without instrumenting or modifying application or system code. LWT is independent of the Guest OS, run-time environments, applications or services being monitored. It requires minimal input from the administrator and introduces minimal performance penalties.

LWT applies to multi-tier applications that have request-response type of interactions, where a client VM makes a request to a server, which performs some computation and responds. The heavier the workload of the client, the more requests we expect it to make. As a result, we expect to see a prominent spike in the server's CPU usage at about the same time we see a spike in the client's CPU usage. By modeling the VMs' CPU usage and clustering them based on the similarity between these models, we are able to predict with high accuracy the dependencies between them.

LWT consists of three stages – *Monitoring, Modelling and Clustering*. The first stage uses a xentop-based monitor to record and extract per VM CPU usage. Next, we estimate an auto-regressive (AR) model for each of the VMs. Finally, we use K-means to cluster the AR models. An autoregressive model is simply a linear regression of the current value of the series against one or more prior values of the series. As we expect communicating VMs to show similar spikes in time, we expect their AR models to be 'close'. Thus, K-means is able to cluster VMs, such that the VMs in one cluster are interdependent. We are able to further improve the results by explicitly perturbing some randomly chosen VMs. Figure 2 shows the LWT system diagram.

The current implementation of LWT has been integrated into the Xen hypervisor running across a small-scale system consisting of 5 physical machines and 31 VMs. The VMs run instances of three different application suites – RUBiS, Hadoop and Iperf. We are able to identify dependencies with an overall accuracy of 97.15%, with a 91.67% accuracy for true positives, and 99.08% accuracy for true negatives.

2 Related Work

We divide related work into four categories: manual, trace-based, middleware-based, and perturbation-based techniques.

Manual Techniques: Management systems like Mercury MAM [3] and Microsoft MOM [4], provide support for application designers to specify the dependency models by maintaining topology maps. However, this approach requires significant manual effort to keep up with the dynamic nature of applications in large systems and is often restricted to a particular set of applications from the same vendor.

Trace-based Techniques: Project5 [7] and WAP5 [17] infer causal paths from black-box network traces. They record messages at each host with both sent and received timestamps. Then Project5 uses an offline nesting and convolution algorithm to infer causal relationships, while WAP5 uses a message-linking algorithm to generate timelines and causal trees. Their approaches differ from our work in that they target the debugging and profiling of individual applications and thus, their primary concern is resolving which incoming packet triggered which outgoing packet. In contrast, we focus on discovering the service dependencies of multi-tier applications running on VMs in real time.

Magpie [10] reconstructs causal paths based on OS-level event tracing. It can automatically extract a system's workload under realistic operating conditions by using event tracing for Windows, built into the Windows OS, to collect thread-level CPU and disk usage information. Magpie, however, requires an application-specific event schema, written by an application expert, to stitch traced information into request patterns.

Orion [14] discovers dependencies for enterprise applications by using time correlation of messages between different services. By 'time correlation' it means that if service A depends on service B, the message delay between A and B should be close to a 'typical' value that exhibits a 'typical' spike in its delay distribution. Similarly, Sherlock [9] calculates an Inference Graph. However, this rule may fail on a virtual machine platform because the 'typical' spike could be easily distorted by various sources of noise, e.g., the domain running A or B might be blocked and may spend uncertain time waiting on the run queue to be scheduled for a CPU.

Gao et al. [15] detect problems in a distributed system by monitoring resources, tracking pair-wise correlations between the monitored parameters, creating a probability model and raising an alarm when the observed data does not comply with the model.

Middleware-based Techniques: Pinpoint [12][13] collects end-to-end traces of client requests traveling through a distributed system by tagging each J2EE call with a unique request-ID. The key to their approach is to use these traces (paths) to enable meaningful automated statistical analysis, e.g., path anomalies and latency profiles can be used to detect system failures. Pinpoint requires all distributed applications to run on homogeneous platforms with the appropriate logging capabilities, but real-life large enterprise datacenter are almost invariably heterogeneous with a plethora of operating systems from different vendors.

Perturbation-based Techniques: Bagchi et al. [8] uncover resource dependencies by taking an active approach, using fault injection. Brown et al. [11] identify cross-domain dependencies by explicitly perturbing system components while monitoring the system's response, e.g., by locking a particular database table to deny the queries from certain component. While we use perturbations to improve our accuracy, we do not rely on it as the sole means of finding dependencies. Pip [16] can obtain high rate of accuracy for extracting causal paths by modifying or at least recompiling the applications.

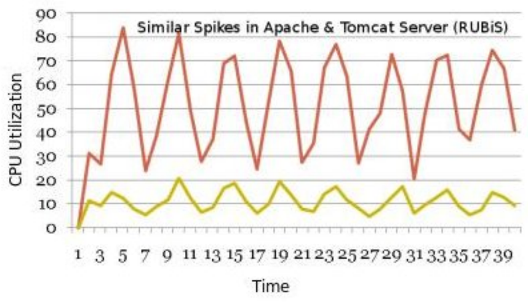


Figure 3: Similar Spikes in CPU Usage of Dependent VMs

3 Design of LWT

In order to successfully achieve the aforementioned goals, our algorithm needs to have the following properties: It should be application and system agnostic, non-intrusive, scalable, resistant to component failures or configuration changes and computationally efficient.

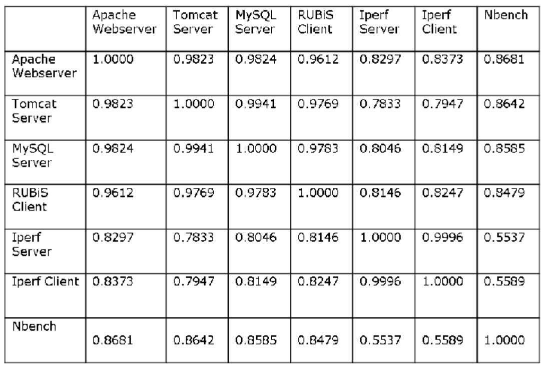
3.1 Intuition & Overview

In a request-response type of architecture, when a client's workload increases, typically, it makes more requests to the server it depends on, and the server's workload, in turn, increases. Hence, we expect to see a prominent *spike* in the CPU utilization of both the client and server simultaneously, proportional to this increase, as illustrated in Figure 3. We cluster the VMs based on the similarity of their respective *spikes* as they appear in time.

We model each VM's CPU utilization as a time series signal and use this information to classify VMs. The algorithm consists of three steps – *Monitoring, Modelling and Clustering*, detailed below. Further, we use active perturbation of randomly selected VMs in order to improve our results.

3.2 Monitoring

We assume that a VM runs either an application in its entirety or a service (component of an application) like a web server. This is a reasonable assumption, given the deployment models in most virtualized datacenters today. The monitoring module records the resource uti-



	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

Figure 4: Inter-VM Correlation - Sampling Period = 3s

lization of VMs per host using `xentop`. The recorded information is then parsed to extract CPU usage details per VM.

3.2.1 Sampling Period

If the sampling period is too small, it increases the amount of computation performed by the algorithm. In the limiting case, it also begins to introduce noise in the readings. Similarly, a very large sampling period will cause us to miss important spikes.

We used sampling periods between 10 milliseconds to 7 seconds for a small set of VMs and used simple correlation to analyze the results. We then selected a cut-off, C, for the correlation coefficient. VMs having a correlation coefficient above this were marked dependent. One such example for a sampling period of 3 seconds is shown in Figure 4, which uses $C = 0.9$. It can be seen from the figure that the Apache Webserver, Tomcat Server, MySQL Server and RUBiS Client show a correlation above the threshold, and therefore are *dependent*; similarly, there exists a dependency between the Iperf client and server, while Nbench, a CPU benchmark is not dependent on any other VMs. Through these experiments, we found that a sampling period of 1 second was an optimal choice.

3.2.2 Sample Size

The sample size required increases gradually as we increase the number of VMs monitored. For our current implementation, with 31 VMs, we have empirically determined that a sample size of 300 (with a sampling period of 1s) is sufficient. Further, since we want to use this approach in a real-time environment, we could monitor continuously and calculate dependencies every 300 or more seconds. Each subsequent result can improve the current notion of dependencies present in the system. This approach can not only accommodate an environment where VMs can be created or destroyed dynamically, but also substantiate or invalidate the dependencies found with each iteration. This also makes the approach less sensitive to the sample size.

3.2.3 Active Perturbation

Perturbation refers to purposely changing some aspects of a VM, such as its available CPU timeslice in order to affect its ability to provide service. Specifically, we set the cap on how much CPU a domain can use, even if the host system has idle CPU cycles. We do this using the `xm` command in Xen to change the amount of "credits" a VM can be given. By perturbing a VM, we expect to see changes in the CPU utilization of dependent VMs. For example, by reducing the cap for a VM, we inhibit its capacity to service requests. Thus, the drop in its CPU utilization will be propagated to dependent VMs, which are waiting to be serviced. By periodically changing this cap on a randomly selected subset of VMs, we introduce additional time-dependent spikes in our sample set.

3.3 Auto-Regressive Modelling

An auto-regressive model for a time series dataset $X = \{x_1, x_2, ..., x_n\}$ is given by a weighted sum of p previous values, where p is the order of the model. The model is given by

$$X_t = c + \sum_p \varphi_i X_{t-i} + \epsilon_t \tag{1}$$

Where $\varphi_1, \varphi_2, ..., \varphi_p$ are the parameters of the model, c is a constant and ε_t is white noise.

By fitting the time series CPU usage of each VM to such a model, we are able to capture how one spike in time is influenced by previous spikes. Although we are not using this model for prediction, it allows us to effectively *summarize* our observations. AR modelling can be performed per host, and takes a finite amount of time for a given order.

3.3.1 Order of AR model

Selecting an order for the model, in this context, is more a choice of *how much* we want to summarize. Since we do not expect this model to be used for prediction , we do not use prediction errors or similar techniques to select the order. This problem is similar to that of selecting the sample size. As the system becomes more complex, we expect the value of p to steadily increase. However, an excessively large value of p will result in overfitting. This trend can be seen in Figure 5, which shows the accuracy of predicting dependencies for different values of p.

For the size of our experimental setup, we find that an order ranging from 40 to 50 works well.

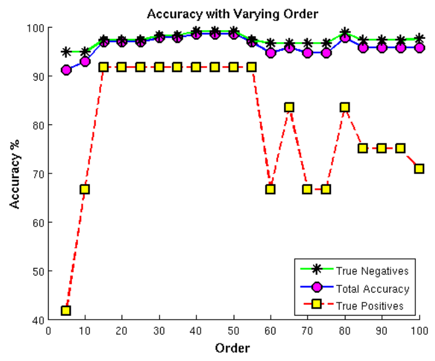


Figure 5: Accuracy with Varying Order of AR models

3.4 Clustering

The final step is to use clustering to group dependent VMs together, based on the distance between their AR models. We use Euclidean distance, so that models with similar coefficients for a particular sample in history (which effectively correspond to spikes in time) will be closer. We use K-means for clustering.

K-means divides the data into a number of clusters=K, where K is provided to the algorithm. K-means does this by selecting K centers or centroids for the data. With each iteration of the algorithm, the points for the centers are improved by decreasing the metric corresponding to intra-cluster distances and increasing that corresponding to inter-cluster distances. The value of K is provided by the administrator.

Although this final step needs to be performed centrally, K-means is very efficient. For a dataset with 1000 samples and 500 real-valued attributes (which corresponds to 1000 VMs with AR models of order 500), K-means finishes computation within a few seconds.

4 Experimental Setup

Our testbed consists of 5 dual core, Dell PowerEdge 1950 compute nodes with Intel Xeon 5150 processors, 4GB of memory, and 80GB hard drives, connected by a gigabit network. We have simulated a cluster with 31 virtual machines, each of which is configured to use 512MB of RAM. We use Xen 3.1.2 as the virtual machine monitor on each host.

The applications and workloads used are described below:

RUBiS [5] is a well-known eBay like benchmark, implementing the core functionality of an auction site: selling, browsing and bidding. We use a servlets-based, four node configuration of RUBiS, consisting of an Apache, Tomcat and MySQL server, along with a fourth client node. One RUBiS instance thus consists of 4 VMs. We use different preexisting workloads provided by RUBiS for different instances of it.

Hadoop MapReduce [1] is a programming model and software framework for writing applications that rapidly process vast amounts of data in parallel on large clusters of compute nodes. We create a single instance of Hadoop using 3 VMs, 1 master and 3 work nodes (one of the VMs is both master and worknode). We use three of the existing sample programs provided with Hadoop - wordcount, randomwriter and sort, to create workloads for Hadoop instances.

Iperf [2] is a commonly used network testing tool that can create TCP and UDP data streams and measure the throughput of a network that is carrying them. We use a client and server functionality based two node configuration of Iperf, where the throughput between the two ends is measured. One Iperf instance consists of 2 VMs.

We use MATLAB to calculate AR models and K-Means.

5 Results

5.1 Performance Evaluation

Figure 5 shows an overview of the results. It shows the total accuracy of predicting dependencies, as well as the breakdown of true positives and negatives upon varying the order of the AR model. As stated in section 3.3.1, it can be seen that an excessively large value of order begins to reduce accuracy due to overfitting. It can also be seen that the method is not extremely sensitive to a particular value of order.

Table 1 shows the breakdown of the results for RUBiS and Hadoop individually as well as for the combined configuration running 3 Hadoop instances, 4 RUBiS instances and 2 Iperf instances for a total of 31 VMs (called *All* workload). It can be seen that we are able to identify, with 100% accuracy, the dependencies in the RUBiS VMs, even without the use of perturbations. This is because an application like RUBiS, which requires a lot of cooperation between the component VMs, is particularly well suited for our approach. It is also the kind of application that will benefit the most from provisioning decisions made by considering the dependencies.

It is more non-intuitive why the approach works well even for Hadoop. The Hadoop Master divides the workload and assigns tasks to the workers (mappers and reducers). After this point, the mappers and reducers communicate via files, i.e. the mappers store intermediate results to files and communicate their location to the master. The master in turn informs the reducers about these locations, so that they can be read and processed. In our setup, all three VMs of a Hadoop instance are workers (one of them is both a master and worker). As a result,

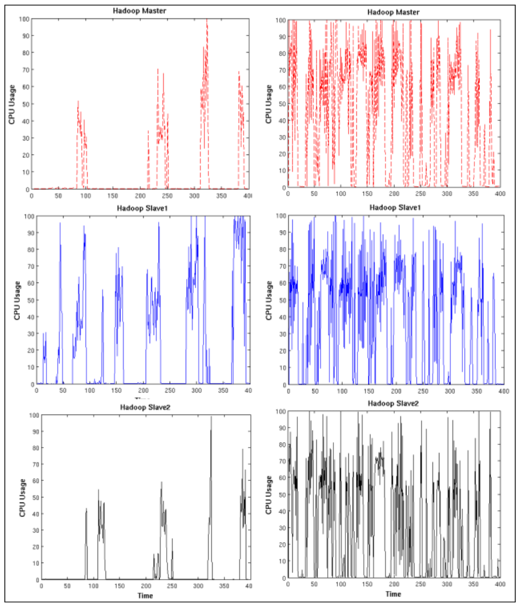


Figure 6: CPU Usage Time Series of Hadoop Instances

there is a fair amount of correlation in their CPU Usage. This can be seen in Figure 6. The figure plots the CPU usage time series of two instances of Hadoop. The left column which belongs to the first instance shows a significantly different pattern than the right column.

Figure 7 shows the effect of perturbations on accuracy. As stated in Section 3.2.3, perturbing a VM by periodically changing its ability to service, introduces additional prominent spikes in the CPU utilization of dependent VMs. Perturbations increase the accuracy of the Hadoop workload by 25% overall (33.33% for true positives and 23.23% for the true negatives). Similarly, for the *All* workload the increase in accuracy is about 2.5%. This seemingly small increase is due to the fact that 96.33% of true negatives have already been identified without the use of perturbations.

Table 1: Results for Identified Dependencies

		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

AR models for two of the application instances are shown in Figure 8, by plotting the coefficients of their previous values in the time series. Models having similar coefficients imply that they have shown similar spikes

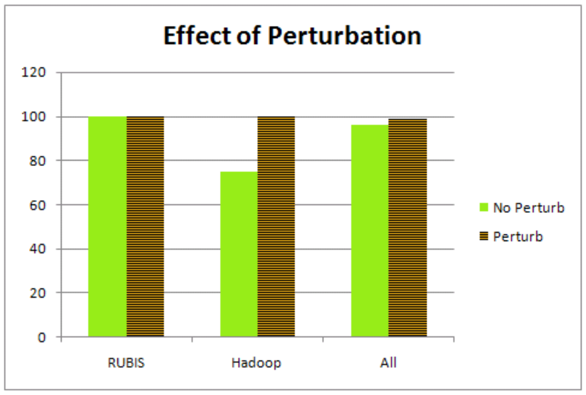


Figure 7: Effect of Perturbations on % Accuracy Rate

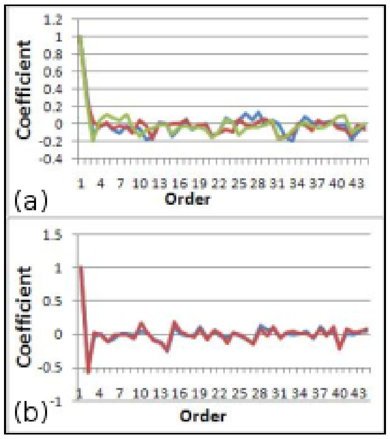


Figure 8: AR Models: (a) RUBiS Model (b) Iperf Model

in time. It can be seen that models from the same application instance are more similar in their coefficients than others, as a result of which they will be clustered together.

Figure 9 shows a scatter plot of the RUBiS VMs, with respect to their 2nd, 3rd and 4th order coefficients. Even in this space, dependent VMs (shown by the same color) are closer to each other.

5.2 Scalability & Time Complexity

The time complexity of the algorithm depends on 3 factors – number of VMs N, choice of order p and choice of sample size W. As we choose p and W before deploying the algorithm on a particular system, we can take their effect to be constant (finite time). The complexity of the first step, finding AR models, is thus, linear in N. Models can be calculated per host and sent to a central machine for the clustering phase. The complexity of k-means is $\Omega(N)$. Although this step is performed centrally, for a dataset with 1000 samples and

500 real-valued attributes (corresponding to 1000 VMs with AR models of order 500), K-means finishes computation within a few seconds. Thus, this algorithm can easily scale in a datacenter of thousands of VMs. To further establish this, we simulated an array of CPU usages for 1200 VMs by copying over an existing utilization array. Using an order $p = 100$, we ran the algorithm over this fictional dataset. Even though we calculated AR models centrally, the total run time for the algorithm was only 1.5 minutes on an Intel core2 duo machine running at 2GHz with 1GB of RAM.

6 Conclusions & Future Work

In this paper we have presented the 'Look Who's Talking' (LWT) set of methods and framework for identifying inter-VM dependencies. The method is non-intrusive, application agnostic, real-time and scalable, and can be easily deployed on a running datacenter with no modifications to VMs. We estimate auto-regressive models for the CPU usage of individual VMs and cluster them based on which models are similar. The current implementation of LWT has been integrated into the Xen hypervisor running across a small-scale prototype datacenter, for which experimental measurements show that it can effectively identify dependencies between VMs with an average of 97.15% overall accuracy rate, identifying almost all true negative dependencies and more than 90% of the true positives.

One of the aspects, as yet to be explored by this work is how the system would react if a large number of VMs depend on a single VM for service. In this case, the characteristic spikes will not be clearly visible. Perturbations will be effective only if deployed on this VM. We would also like to automate the discovery of parameters like the order of the AR models and sample size.

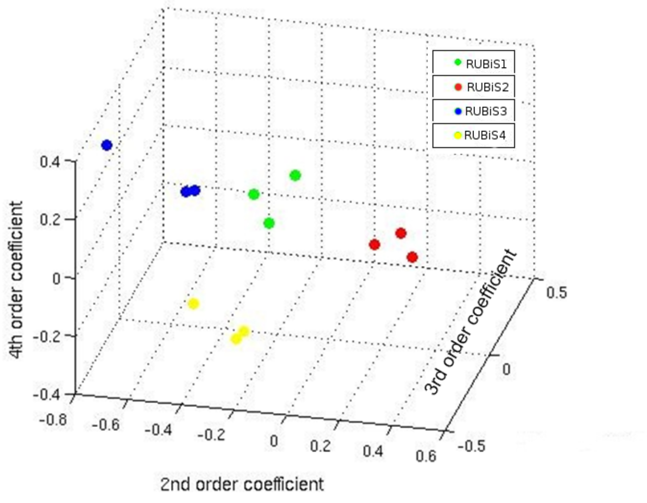


Figure 9: Visualization of Models in Space

References

[1] Hadoop. `http://hadoop.apache.org/`.

[2] Iperf. `http://sourceforge.net/projects/iperf`.

[3] Mercury mam. `http://www.mercury.com/us/products/business-availability-center/application-mapping`.

[4] Microsoft mom. `http://technet.microsoft.com/en-us/systemcenter/om/bb498244.aspx`.

[5] Rice university bidding system. `http://rubis.ow2.org/index.html`.

[6] Virtual computing environment. `http://www.vmware.com/company/news/releases/virtual-computing-environment.html`.

[7] AGUILERA, M. K., MOGUL, J. C., WIENER, J. L., REYNOLDS, P., AND MUTHITACHAROEN, A. Performance debugging for distributed systems of black boxes. In *SOSP* (2003), pp. 74–89.

[8] BAGCHI, S., KAR, G., AND HELLERSTEIN, J. Dependency analysis in distributed systems using fault injection application to problem determination in an e-commerce environment. In *12th Intl Workshop on Distributed Systems Operations and Management* (2001).

[9] BAHL, P., CHANDRA, R., GREENBERG, A., KANDULA, S., MALTZ, D. A., AND ZHANG, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. *SIGCOMM Comput. Commun. Rev. 37*, 4 (2007), 13–24.

[10] BARHAM, P., DONNELLY, A., ISAACS, R., AND MORTIER, R. Using magpie for request extraction and workload modelling. In *OSDI* (2004), pp. 259–272.

[11] BROWN, A., KAR, G., AND KELLER, A. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *In Seventh IFIP/IEEE International Symposium on Integrated Network Management* (2001).

[12] CHEN, M. Y., KICIMAN, E., ACCARDI, A., FOX, A., AND BREWER, E. A. Using runtime paths for macroanalysis. In *HotOS* (2003), pp. 79–84.

[13] CHEN, M. Y., KICIMAN, E., FRATKIN, E., FOX, A., AND BREWER, E. A. Pinpoint: Problem determination in large, dynamic internet services. In *DSN* (2002), pp. 595–604.

[14] CHEN, X., ZHANG, M., MAO, Z. M., AND BAHL, P. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI* (2008), R. Draves and R. van Renesse, Eds., USENIX Association, pp. 117–130.

[15] GAO, J., JIANG, G., CHEN, H., AND HAN, J. Modeling probabilistic measurement correlations for problem determination in large-scale distributed systems. In *ICDCS '09: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 623–630.

[16] REYNOLDS, P., KILLIAN, C. E., WIENER, J. L., MOGUL, J. C., SHAH, M. A., AND VAHDAT, A. Pip: Detecting the unexpected in distributed systems. In *NSDI* (2006).

[17] REYNOLDS, P., WIENER, J. L., MOGUL, J. C., AGUILERA, M. K., AND VAHDAT, A. Wap5: black-box performance debugging for wide-area systems. In *WWW '06: Proceedings of the 15th international conference on World Wide Web* (New York, NY, USA, 2006), ACM, pp. 347–356.