Net-Cohort: Detecting and Managing VM Ensembles in Virtualized Data Centers

Liting Hu[1], Karsten Schwan[1], Ajay Gulati[2], Junjie Zhang[1], Chengwei Wang[1]

[1]College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{foxting, schwan, jjzhang, flinter}@cc.gatech.edu

[2]Resource Management Team
VMware, Inc.
Palo Alto, CA 94304
agulati@vmware.com

ABSTRACT

Bi-section bandwidth is a critical resource in today's data centers because of the high cost and limited bandwidth of higher-level network switches and routers. This problem is aggravated in virtualized environments where a set of virtual machines, jointly implementing some service, may run across multiple L2 hops. Since data center administrators typically do not have visibility into such sets of communicating VMs, this can cause inter-VM traffic to traverse bottlenecked network paths. To address this problem, we present `Net-Cohort', which offers lightweight system-level techniques to (1) discover VM ensembles and (2) collect information about intra-ensemble VM interactions. Net-Cohort can dynamically identify ensembles to manipulate entire services/applications rather than individual VMs, and to support VM placement engines in co-locating communicating VMs in order to reduce the consumption of bi-section bandwidth. An implementation of Net-Cohort on a Xen-based system with 15 hosts and 225 VMs shows that its methods can detect VM ensembles at low cost and with about 90.0% accuracy. Placements based on ensemble information provided by Net-Cohort can result in an up to 385% improvement in application throughput for a RUBiS instance, a 56.4% improvement in application throughput for a Hadoop instance, and a 12.76 times improvement in quality of service for a SIPp instance.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management – *scheduling*; D.4.7 [**Operating Systems**]: Organization and Design – *distributed systems*.

General Terms

Algorithms, Management, Design.

Keywords

Virtualization, Clustering, Dependency Analysis.

1. INTRODUCTION

Virtualization is being deployed in data centers at a rapid pace to consolidate workloads for improved server utilization, for ease of provisioning, configuration management, and more generally, for flexible use of data center resources. A typical application running in a virtualized environment consists of a set of virtual machines (VMs) – a VM ensemble – that cooperate and communicate to jointly provide a certain service or accomplish a task. A multi-tier web application, for instance, may be structured as an ensemble with certain VMs implementing its front end service, other VMs running application servers, and backend VMs running databases or network file systems.

VM ensembles can be configured and mapped to data center machines to scale throughput by partitioning tasks across multiple machines, to obtain high availability by mapping VMs to different nodes or racks, or to improve power consumption by minimizing the number of machines used by an ensemble, while still meeting performance and reliability requirements [18].

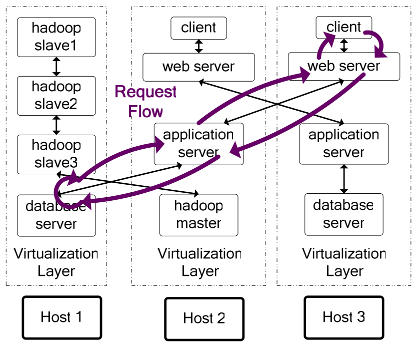


Figure 1. VM ensemble consisting of a multi-tier application mapped across multiple hosts in a data center.

Figure 1 shows one such configuration for a multi-tier e-commerce web application represented by RUBiS. In this case, client requests arrive at the VM running the web server front end and are then forwarded to one of the VMs running application servers, which in turn may request data from a backend VM hosting a database.

Bi-section bandwidth of the network infrastructure is a critical, scarce, and expensive resource in data centers today. Recent studies [11] [20] [21] have shown that servers in different racks have to share the up-links from top of rack switches (ToRs). Since these are typically 5:1 to 20:1 oversubscribed, this can result in a worst-case available bi-section bandwidth as low as 125Mbps [17]. Furthermore, higher level switches in the network topology cost much more, due to the amount of network bandwidth and numbers of ports they have to support.

Limited bi-section bandwidth places constraints on the mapping of VM ensembles to underlying hosts. As illustrated in Figure 2, an ensemble of frequently communicating, 'chatty' VMs is placed across multiple racks. Such a placement can negatively affect the services provided by the ensemble [26]. First, as the shared up-links from ToRs become saturated, intra-ensemble communications may be delayed. Such delays can be further exacerbated by message retransmissions due to time-outs. Second, the use of scarce, shared bandwidth can affect other services and ensembles, as evident in applications like Hadoop that experience slowdown due to file system-level data reorganization. This is also demonstrated in one of our experiments, where a RUBiS benchmark experiences a 79.4% performance loss in application throughput when placed across a bandwidth-constrained set of machines (see Section 6). Finally, a link failure can cause severe imbalances across paths and may require relocation of some of the VMs in order to reduce over-subscription.

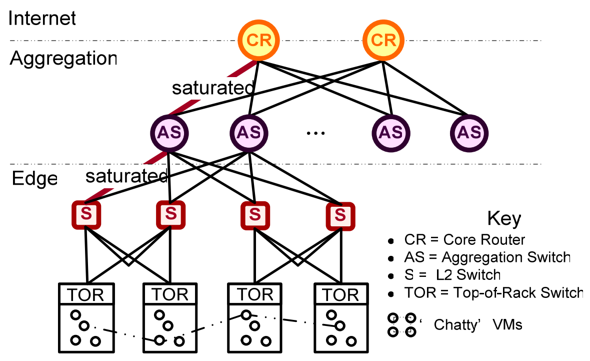


Figure 2. Example of shared up-links from TORs crash, causing performance degradation for many VMs.

This paper presents 'Net-Cohort', a lightweight system that (1) continuously monitors a system to identify potential VM ensembles, (2) assesses the degree of 'chattiness' among the VMs in these potential ensembles, thereby (3) enabling optimized VM placement to reduce the stress on bi-section bandwidth of the data center's network. 'Net-Cohort' has the following unique properties:

- *scalable* – it accurately identifies 'chatty' VMs using commonly available runtime statistics and a two-step method for increased precision when needed;
- *actionable* – insights derived from running Net-Cohort can help management software better co-locate VM ensembles on underlying hosts;
- *privacy-preserving* – its black-box methods do not require any VM (guest-OS) level changes or any information about the VM ensemble being run from the user.

Discovering VM ensembles and their inter-VM dependencies is quite challenging. A naïve method that continuously gathers statistics about all communicating VM-pairs is prohibitively expensive. First, it would require introspection of all packets sent and received by the VMs; this would induce notable CPU overheads and additional per packet latencies of the tens of microseconds. Second, additional memory resources would be required to maintain statistics for every pair of IP addresses.

Net-Cohort uses a two-step approach to limit runtime overheads in terms of metric collection and per packet analysis. The first step acquires VM-level statistics commonly available in virtualized systems, such as the total numbers of packet in/out over time. It then computes the correlation coefficients among these statistics and divides the corresponding VMs into subsets (also called ensembles) using correlation values and a hierarchical clustering algorithm.

The second step uses a statistical packet sniffer only on the VMs identified as members of a misplaced ensemble, i.e., an ensemble with VMs placed across remote racks. The packet sniffer maintains information about outgoing packets, their destination IP addresses and corresponding counters, which are then used to determine the actual communication intensity among VMs. To optimize memory consumption, only the top-k destinations are tracked in an online manner using the statistical algorithm proposed by Lukasz and David in [16]. Finally, this information about VM level communications is used to drive new VM placement decisions.

Determining a new placement of VMs to physical servers is similar to multi-dimensional bin-packing problem. Placement requires evaluation of multiple criteria such as balancing of CPU, memory and I/O resources on each host. Existing solutions like VMware DRS, use weighted mechanisms to combine the standard deviation across multiple dimensions. Different policies can also influence the placement. For example, power savings would give priority to consolidating VMs on fewer servers whereas load-balancing would redistribute VMs across all servers. Net-Cohort can supplement any placement system by providing network communication cost as another dimension. However, designing a placement solution just based on network communication would not be very useful and designing a complete solution is out of scope of this paper. As suggested in the Section 4, Net-Cohort can suggest soft or hard affinity rules between VMs and the placement engine (e.g. VMware DRS) can enforce them during load-balancing.

We have implemented Net-Cohort on Xen hypervisor, and evaluated its effectiveness on a virtualized infrastructure consisting of 15 hosts and 225 VMs. These VMs run a diverse mix of business, web, Internet services, and batch workloads. Experimental results show that Net-Cohort can identify VM ensembles with 90.0% accuracy, and improved placements due to its use can increase application performance in terms of throughput and latency. In particular, we observe an up to 385% improvement in application throughput for a RUBiS instance, a 56.4% improvement in application throughput for a Hadoop instance, and a 12.76 times improvement in quality of service for a SIPp instance.

The remainder of this paper is organized as follows. Section 2 discusses background and related work. Section 3 describes the Net-Cohort design and implementation. Section 4 discusses the support and integration with VM placement engines. Sections 5 and 6 present the experimental setup and performance evaluation, respectively. We conclude with some directions for future work in Section 7.

2. BACKGROUND AND RELATED WORK

We first explain the dominant design pattern for today's data centers [2] and why an inappropriate placement of VMs on data center machines can incur substantial performance penalties. We then discuss the related literature.

2.1 Data Center Background

As shown in Figure 2, data center networks are based on a proven layered approach, including a layer of servers in racks at the bottom (access layer), a layer of aggregation 10 Gigabit Ethernet switches at the middle (aggregation layer), and a layer of core routers at the top (core layer). There are typically 20 to 40 servers per rack, each singly connected to a Top of Rack (ToR) switch with a 1 Gbps link. ToRs connect to End of Row (EoR) switches via 1-4 of the available 10 GigE uplinks, and these switches manage traffic into and out of the rack. At the top of the hierarchy, core routers carry traffic between aggregation switches and manage traffic into and out of the data center.

As traffic moves up through the layers of switches and routers, the over-subscription ratio, which is the ratio of the allocated bandwidth per host to the worst-case guaranteed bandwidth per host, increases rapidly. For example, for servers in the same rack, they can communicate at the full rate of their interfaces (e.g., 1Gbps) with 1:1 over-subscription ratio. Unfortunately, servers in different racks have to share the up-links from ToRs, which are typically 5:1 to 20:1 oversubscribed, resulting in 125Mbps as the worst-case available bi-section bandwidth [17].

Network latencies may not vary much, but the bandwidth available within a rack, across racks, and across rows can vary substantially. Therefore, inappropriate placement of VMs can have dire consequences. An example is the placement of heavily communicating VMs across multiple racks, thereby consuming the bandwidth available to a QoS-sensitive VM ensemble. Based on anecdotal evidence, users deploy multiple VMs in a public cloud to finally find a group with low ping latency between them and then hold on to them. Such scenarios motivate us to develop Net-Cohort's runtime methods to identify VM ensembles, as once identified, they can better place VMs onto data center machines.

2.2 Related Work

We classify related literature into four different categories: manual techniques, trace-based and middleware-based techniques, and techniques using explicit perturbation.

2.2.1 Manual Techniques

Some sophisticated network management systems, e.g., Mercury MAM [3] and Microsoft MOM [4], rely on application designers or owners to specify dependency models. This restricts these approaches to particular applications or vendors and requires significant updates or changes when applications evolve. This is not very practical both for public clouds like Amazon EC2 or for private cloud deployments to run the IT for large enterprises like those reported in a survey conducted by the Wall Street Journal'08, which states that a single company, Citigroup, operates over 10,000 line-of-business applications [15].

2.2.2 Trace-based Techniques

Project5 [9] and WAP5 [23] infer causal path patterns from offline network traces, using messages at hosts recorded with both sent and received timestamps. Project5 infers causal relationships between two message streams by computing their cross correlation. WAP5 generates timelines and causal trees, based on the assumption that causal delays follow an exponential distribution. The project's purpose is to isolate performance bottlenecks, e.g., to detect which nodes are sources of latency. Their primary concern, therefore, is to resolve which incoming message triggers which outgoing message. In contrast, Net-Cohort operates at a larger scale and requires less information about the underlying system.

E2Eprof [8] reconstructs causal paths based on kernel-level network tracing. Compared to Net-Cohort, E2EProf has higher runtime overheads due to capturing the end-to end latencies of all requests in multi-tier systems and applying cross correlation analysis to all network flows.

Orion [15] discovers dependencies for enterprise applications by using the 'time correlation' of messages between different services, meaning that if service A depends on service B, the message delay between A and B should be close to a "typical" value. Applying this rule to VM platforms may be difficult, because a "typical" spike could be distorted by noise, e.g., the domain running service A or B may lose its processor and spend some uncertain amount of time waiting to be scheduled.

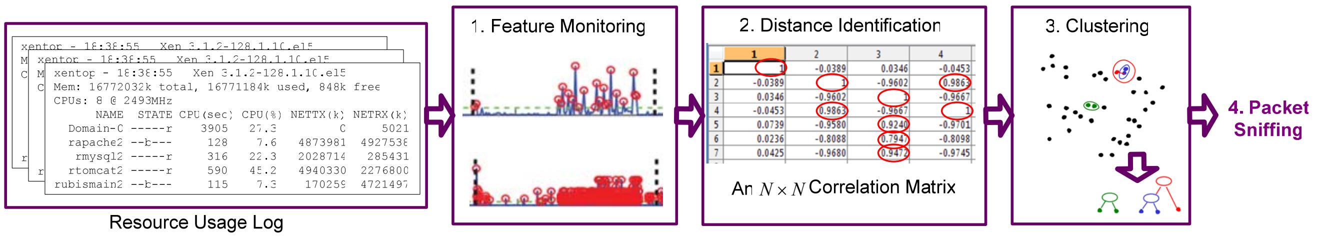
Our earlier workshop paper [10], called LWT, is not sufficiently accurate or flexible: they are based only on CPU metrics; their use of k-means clustering requires parameters settings to be customized to the applications being run.

Pinpoint [14] collects end-to-end traces of client requests travelling through a distributed system, by tagging each J2EE call with a unique request-ID. These traces enable automated statistical analyses. Pinpoint requires all distributed applications to run on homogeneous platforms with logging capabilities, but real-life large enterprise data centers are almost heterogeneous with a plethora of operating systems from different vendors.

2.2.3 Middleware-Based Techniques

vPath [24] provides path discovery by monitoring and recording thread and network activities at runtime, such as which thread performs a *send* or *recv* system call over certain TCP connection. vPath can be implemented in either the OS kernel or a virtual machine monitor (VMM). Although the implementation is agnostic to user-space code, it requires changes to the VMM code and the guest OS.

Aurora [1] is targeted at flow-based network traffic analysis for large networks to provide anomaly and virus detection/mitigation, BGP/OSPF/RIP monitoring, and traffic network maps. It discovers communication dependencies among servers through detailed network traffic reports from NetFlow, which a network protocol developed by Cisco Systems. However, it requires an installation of a netflow collector and works specifically on Cisco routers. The cost and overhead of generating the traffic report can be quite high in large datacenters.



```
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
```

	1	2	3	4
1	1	-0.0389	0.0346	-0.0453
2	-0.0389	1	-0.9602	0.9863
3	0.0346	-0.9602	1	-0.9667
4	-0.0453	0.9863	-0.9667	1
5	0.0739	-0.9580	0.9240	-0.9701
6	0.0236	-0.8088	0.7947	-0.8098
7	0.0425	-0.9680	0.9472	-0.9745

Figure 3. Workflow of Net-Cohort.

2.2.4 *Perturbation-Based Techniques*

Resource dependencies are uncovered using an active approach in [12] – using fault injection. In [13], cross-domain dependencies are identified 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. Pip [22] can obtain a high rate of accuracy for extracting causal paths by modifying, or at least recompiling the applications. The generality of all these approaches is limited by that fact that they require expert knowledge about the systems and applications being evaluated.

3. NET-COHORT DESIGN

Net-Cohort's uses a three-step approach to discover VM ensembles and make use of them. These include (1) monitoring of some basic statistics, (2) computing potential ensembles using correlation techniques and creating hierarchical clusters, and (3) collecting more precise communication statistics between the VMs in each ensemble. Figure 3 shows these steps at a high level, with additional detail presented next.

3.1 Data Collection

The basic monitor module captures the following universally available system-level metrics about each VM:

- *CPU*: CPU usage per second in percentage terms (%)
- *PacketOut*: Packets transmitted per second (KByte/sec),
- *PacketIn*: Packets received per second (KByte/sec)

These periodic measurements result in three time series signals per VM. Before explaining the actual analysis being applied, we illustrate the utility of taking these measurements with a simple example.

Multi-tier applications (e.g., RUBiS) typically use a request-response architecture, in which a client node sends a request to the front end (e.g., Apache), which does load balancing and assigns the work to an appropriate server (e.g., Tomcat) running the application logic. The application logic services the request by querying the backend (e.g., a database server like MySQL) to produce the necessary output, and sending the response back to the client.

Therefore, given the nature of multitier applications, we can expect correlations between one or more of "*CPU*", "*PacketOut*", and "*PacketIn*" statistics for interacting nodes. Figure 4 shows these measurements for Client, Apache, Tomcat and MySQL VMs for one instance of RUBiS. Here we find that the "spikes" and "valleys" of their packet flows consistently occur together. The reason is that an instant rise of CPU usage or packet flow rate in one VM directly or indirectly triggers activities in other VMs, thereby creating the correlations among these statistics.

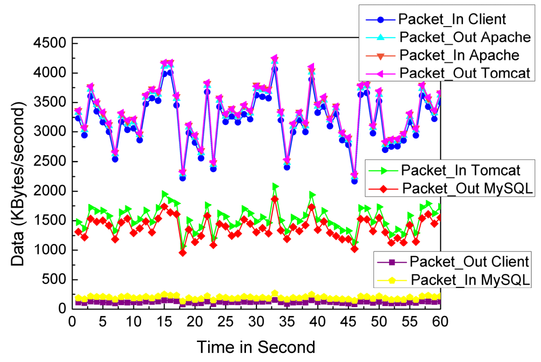


Figure 4. Example of a multi-tier RUBiS application showing correlation on packet flowrate.

Although "*PacketOut*" and "*PacketIn*" are more intuitive metrics to do correlation analysis among communicating VMs, we also used "*CPU*" metric for the analysis because it provides extra information in certain scenarios. For example, a bunch of VMs doing many-to-many but infrequent large data exchanges. It gets even harder to detect these using only packet-based metrics if the interval between exchanges is larger than the sampling window. In these cases, CPU correlation provided better results as compared to others. We found one such example during our evaluation as well (see Section 6.1, Figure 9).

3.2 Distance Identification

For each pair of VMs in the data center, we calculate the correlation coefficient value, denoted as corr(V_i, V_j), to identify the dependency strength between them. In statistics, the correlation coefficient indicates the strength and direction of a linear relationship between two random variables. We choose the pearson product-moment correlation coefficient (PMCC) to measure the degree of correlation, giving a value between +1 and −1 inclusive.

Let $X = \{X_1, X_2, ..., X_n\}$ and $Y = \{Y_1, Y_2, ..., Y_n\}$ be the vectors of two random variables, then the strength of the dependence between X and Y is:

$$corr(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(Xi - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \quad \ldots\ldots \quad \ldots.. (1)$$

Here $\bar{X}$ *and* $\bar{Y}$ denote the sample means of X and Y, respectively. We expect to see a large, positive correlation coefficient between certain statistics of the interacting VMs.

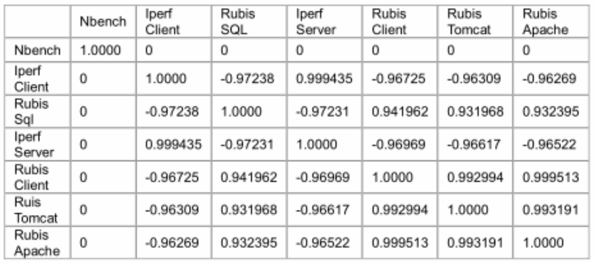
3.3 Compute Pair-wise Distance among VMs

Let w be the size of observation window in seconds. Let $V = \{VM_1, VM_2, ..., VM_N\}$ be the set of all VMs in the data center. For each observation window, let $CPU_i = \{CPU_1, CPU_2, ..., CPU_w\}_i$ denote the set of CPU usage readings collected for VM_i; let $PacketOut_i = \{PacketOut_1, PacketOut_2, ..., PacketOut_w\}_i$ be the set of *PacketOut* readings collected for VM_i; let $PacketIn_i = \{PacketIn_1, PacketIn_2, ..., PacketIn_w\}_i$ be the set of *PacketIn* readings collected for VM_i.

The readings interval is few seconds. For each pair of VMs (denoted as V_i, V_j), we calculate the correlation coefficients for the following three combinations of statistics using Equation (1):

- $corr_1 = corr(CPU_i, CPU_j)$
- $corr_2 = corr(PacketOut_i, PacketIn_j)$
- $corr_3 = corr(PacketIn_i, PacketOut_j)$

Next as illustrated in Figure 5, a $N \times N$ correlation matrix for the set of all VMs is generated, in which the correlation coefficient between V_i and V_j, (denoted as $corr(V_i, V_j)$) is set as the maximum value of the three combinations, $Max(corr_1, corr_2, corr_3)$. The above process is repeated for each observation window. Finally, we have ρ $N \times N$ correlation matrices (ρ refers to the number of observation windows).



	Nbench	Iperf Client	Rubis SQL	Iperf Server	Rubis Client	Rubis Tomcat	Rubis Apache
Nbench	1.0000	0	0	0	0	0	0
Iperf Client	0	1.0000	-0.97238	0.999435	-0.96725	-0.96309	-0.96269
Rubis Sql	0	-0.97238	1.0000	-0.97231	0.941962	0.931968	0.932395
Iperf Server	0	0.999435	-0.97231	1.0000	-0.96969	-0.96617	-0.96522
Rubis Client	0	-0.96725	0.941962	-0.96969	1.0000	0.992994	0.999513
Ruis Tomcat	0	-0.96309	0.931968	-0.96617	0.992994	1.0000	0.993191
Rubis Apache	0	-0.96269	0.932395	-0.96522	0.999513	0.993191	1.0000

Figure 5. Example of a $N \times N$ correlation matrix.

Given the ρ observation windows, we need to handle the noise and outliers in incoming data. Two negative situations have to be excluded: (1) unrelated VMs showing "similar" resource usage features lasting for μ observation windows ($\mu << \rho$); (2) two correlated VMs showing "diverse" resource usage features lasting for μ observation windows ($\mu << \rho$). To avoid these abnormal samples from affecting our result, we calculate the average value for $corr(V_i, V_j)$ based on ρ windows. We further define the distance between V_i and V_j as:

$$Dist(V_i, V_j) \begin{cases} 1/corr(V_i, V_j) & \text{if } corr(V_i, V_j) > 0 \\ \infty & \text{if } corr(V_i, V_j) \le 0 \end{cases}$$

The distance indicates the strength of dependency between two VMs, meaning that the closer the distance, the stronger the correlation. We then calculate a $N \times N$ distance matrix for all VMs, which will be used in the next step.

3.4 Hierarchical Clustering

Given a set of N VMs with a $N \times N$ distance matrix, Net-Cohort uses an unobtrusive black-box approach to cluster them into ensembles. The black-box clustering requires only VM-level's external observations like CPU usage over time, packet transmitted over time, to indicate which VMs may be interacting with each other. Such observations are commonly available from any data center's monitoring system or hypervisors and thus collecting the source data is easy and lightweight.

Two commonly used clustering algorithms are hierarchical clustering and *k*-means clustering. Net-cohort uses hierarchical clustering for the following reasons:

- Hierarchical clustering does not require the number of clusters in advance.
- It works well with both globular and non-globular clusters, while *k*-means fails to handle non-globular data.
- *k*-means clustering is sensitive to initial centroids. If the user does not have adequate knowledge about the data set, this may lead to erroneous results.

The process of hierarchical clustering is shown in Figure 6:

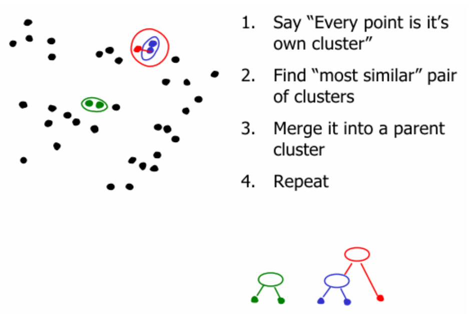

1. Say "Every point is it's own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat

Figure 6. Example of hierarchical clustering.

- Step 1: initially assign each VM to a cluster, so that there are *N* initial cluster for *N* VMs.
- Step 2: find the closest (most similar) pair of clusters and merge them into a single cluster.
- Step 3: compute distances (similarities) between the new cluster and each of the old clusters.
- Step 4: repeat Steps 2 and Step 3 until all items are clustered into a single cluster of size *N*.

Concerning Step 4, of course, there is no point in having all *N* items grouped into a single cluster, but doing so results in the construction of the complete hierarchical tree, which can be used to obtain *k* clusters by just cutting its *k-1* longest links. *K* can be based on the number racks in the datacenter, or it can be chosen to make the inter-cluster distance is less than a certain threshold.

3.5 Statistical Packet Sniffing

Based on the ensembles information from Section 3.3 and the initial known VM/PM mappings, Net-Cohort detects the ensemble whose components might be misplaced, e.g., a RUBiS ensemble but its components (Apache, Tomcat, Database) are placed across remote racks. Then packet sniffing is enabled only on such misplaced VMs to determine the actual communication intensity among them. Calculating the communication intensity is important because a bunch of VMs that do frequent, synchronized exchanges would appear to be an ensemble by the first step, but their communication volume transferred might be trivial and will have little impact on bi-section bandwidth.

Net-cohort uses a lightweight packet sniffer inside domain0 to log the destination IPs and communication frequency for a VM.

Note that not all VMs with high correlation are suitable for being placed close to each other. Informally, the VMs whose communication exceed a certain threshold and occur for a "sustained" period are better candidates for co-location. After all, there is no need to co-locate VMs that have only lightweight "ping pong" packet communications. However, the hierarchical clustering tree lacks information about how much data has been exchanged between dependent VMs. By contrast, the packet sniffer can provide a better estimate of data transmission.

In order to make the data collection more efficient, only *top-K* destinations in terms of network communication are collected. Consider a data center with large number of VMs, the overhead for maintaining data structures of all distinct IP addresses might be costly. We therefore adapt well-known techniques from data mining, as proposed by Lukasz and David in [16], to compute *top-K* destination IP addresses and an approximate packet count. This algorithm identifies frequent items in sliding windows defined over real-time packet streams with limited memory. More details can be found in [16]. Packet sniffer is not completely transparent and introduces some CPU and memory overhead (see Section 6.3), so we only apply it to the misplaced ensembles and run it for a short period.

4. SUPPORT FOR PLACEMENT ENGINE

Based on VM ensemble discovery and fine-grained statistics, Net-Cohort can supplement any VM placement engine to identify collocation opportunities for VM that are communicating heavily.

A VM placement engine is responsible for managing the mappings of VMs onto physical machines (PMs) in accordance with criteria specified by users or administrators. For example, if the optimization criterion is to minimize power usage, the VM placement engine might consolidate VMs onto fewer hosts and power off unneeded hosts during periods of low resource utilization (e.g., VMware DPM); if the optimization criterion is to achieve high levels of QoS, VM placement engine might perform load-balancing of VMs between physical hosts when specific thresholds are exceeded, such as transactions per second, CPU utilization, etc. (e.g., Microsoft's Performance and Resource Optimization (PRO) [5], VMware DRS and DPM [7]).

With Net-Cohort, a VM placement engine can be enhanced to consider VM level communication in its decisions. One way for Net-Cohort to interact with a placement engine is to provide it with a list of candidates for VM movement, ranked by the potential benefits accrued in terms of reduced use of network bi-section bandwidth. The engine, then, considers those facts along with other migration criteria, such as load balancing and VM migration costs, to make migration decisions. Another alternative is to set soft affinity rules among VMs that are communicating heavily, which is then considered by the placement engine when it decides on VM relocations. If the VM level communications are known in advance or from historical records, the VM placement engine can co-locate them from the very beginning.

Regardless of how an engine leverages Net-Cohort's information, however, it is the engine's task to resolve conflicts between different rules and the movements they suggest. Placement engine design and details are beyond the scope of this paper, for our analysis we assume that we have enough resources in terms of CPU and memory to do collocation of VMs based on their communication intensity and there is not interfering rule regarding separating them for higher availability.

5. EXPERIMENTAL SETUP

This section validates Net-Cohort with experiments performed with representative applications in a virtualized infrastructure.

5.1 Testbed

Experiments are run on 15 dual-core dual-socket servers, each of which has two Intel Xeon 5150 processors, 16GB of memory, and 80GB hard drives. The 15 servers are distributed across 4 edge switches, with 4, 4, 4, and 3 servers/switch. All switch and NIC ports run at 1Gbps. Switches are connected to each other, with an oversubscription ratio of at most 4:1.

Applications are embedded in a total of 225 virtual machines, each of which is configured to use 128MB of RAM. Xen 3.1.2 is used as the virtual machine monitor on each host, and the host kernel for XenoLinux is a modified version of Linux 2.6.18.

5.2 Workload and Metrics

Experiments employ 6 instances of RUBiS (24 VMs), 3 instances of Hadoop (48 VMs), 6 instances of Iperf (12 VMs), 6 instances of N-bench (6 VMs), 6 instances of SIPp (12 VMs) and 3 instance of MalGen (123VMs), resulting in a total of 225 VMs running a mix of business, internet services, and batch workloads.

RUBiS is an eBay-like benchmark. We use a PHP-based configuration of RUBiS, with a web server front end (Apache) and an application server (Tomcat) connected to a database backend (MySQL). Workload generation uses a fourth client VM.

SIPp is a traffic generator for the SIP protocol. It can establish actual client and server sessions and initiate/release thousands of calls with given rate. We use call rates (calls per second) starting from 800, increased by 10 every second, and a maximum rate of 3000, with total calls set to 1000K.

Hadoop is a Map/Reduce-like framework in which the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster.

N-bench is a CPU benchmark measuring CPU performance, such as integer operations and floating point arithmetic.

Iperf is a commonly used network workload generation tool. We continuously run Iperf pairs to generate interference traffic, thereby causing the bandwidth bottleneck.

MalGen is a set of scripts that generate a large, distributed data set across multiple nodes in a cluster. One MalGen instance consists of one seed and a dozen workers.

6. EXPERIMENTAL EVALUATION

We first evaluate the basic functionality of Net-cohort (Section 6.1) in finding the right set of ensembles. We then evaluate the benefit of these findings for better VM placement on hosts through the comparison of application throughput and response time between initial VM placement and final VM placement (Section 6.2). Finally we evaluate the overhead, if any, induced by Net-Cohort (Section 6.3).

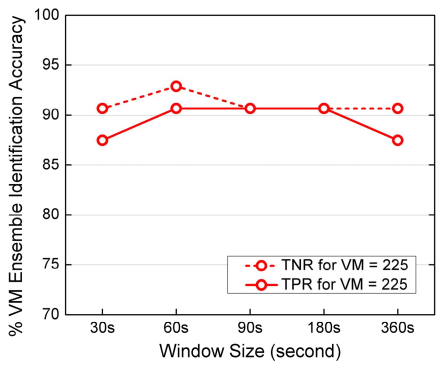


Figure 7. VM ensemble identification accuracy (in percentage) over 225 VMs with an increasing window size.

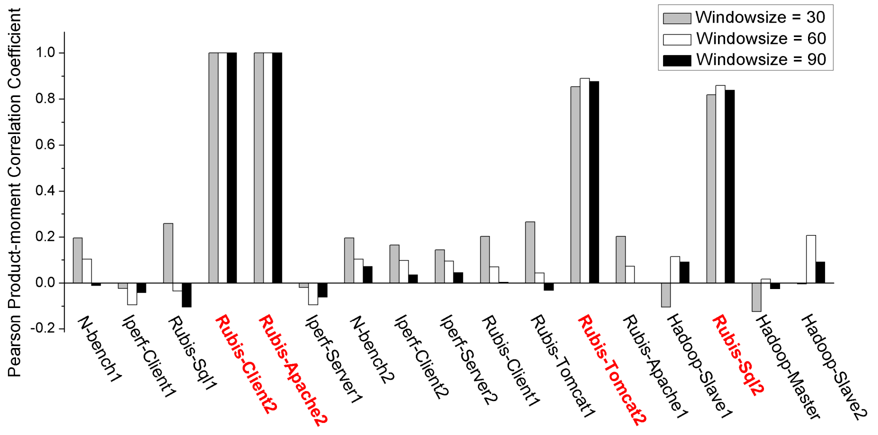


Figure 8 . Intra-cluster/Inter-cluster correlation coefficient among RUBiS2-Client and other VMs.

6.1 Net-Cohort's Functionality Evaluation

The following metrics are used to evaluate Net-Cohort's functionality: (1) *true positive rate* (TPR) and *true negative rate* (TNR). TPR is defined as the ratio of the VMs that have been grouped into the right ensembles to the total set of VMs, denoted as $|V_{TP}|/|V|$. TNR is defined as the ratio of the VMs that have not been grouped into the wrong ensemble to the total set of VMs, denoted as $|V_{\mathrm{NP}}|/|V|$. (2) The *intra-cluster correlation coefficient* is defined as the representative correlation coefficient value for pairs of VMs within one particular ensemble. The (3) *hierarchical tree structure* is comprised of the VM ensembles determined by the hierarchical clustering algorithm.

Figure 7 presents the accuracy of VM ensemble identification by Net-Cohort over 225 VMs with an increasing window size. The TPRs are 87.5%, 90.0%, 90.0%, 90.0%, and 87.5% for window sizes 30 seconds, 60 seconds, 90 seconds, 180 seconds, and 360 seconds, respectively. The TNRs are 90.0%, 92.5%, 90.0%, 90.0%, and 90.0% for window sizes 30 seconds, 60 seconds, 90 seconds, 180 seconds, and 360 seconds, respectively.

The results shows that: (1) without relying on any knowledge of the configuration of the test system, Net-Cohort can correctly classify around 90% of virtual machines; (2) there is not much difference between the results for different window sizes, indicating that Net-Cohort is relatively insensitive to the size of the observation window and thus user-friendly for system administrators to set its parameters.

To better understand the above results, we look at correlation coefficients among various VMs within and across ensembles that are calculated by Net-Cohort. For the sake of clarity, we have selected a few representative VMs and presented their correlation with a random subset of all the 225 VMs. The subset includes 7 instances ((N-bench$_1$, N-bench$_2$, Iperf$_1$, Iperf$_2$, RUBiS$_1$, RUBiS$_2$ and Hadoop) out of all 30 instances, where N-bench$_1$ denotes the first instance of N-bench, Iperf$_2$ denotes the second instance of Iperf, etc. The figures show the intra-cluster/inter-cluster correlation coefficient for a representative VM and 17 VMs, where X-axis represents certain VM's name and Y-axis represents PMCC value between the representative VM and that VM. We find that the VMs belonging to the same application have relatively strong correlations and thus shorter distances.

Figure 8 shows the correlation coefficient between among RUBiS$_2$-Client and RUBiS$_2$-Apache, RUBiS$_2$-Tomcat, RUBiS$_2$-Sql. Note that the *intra-cluster correlation coefficients* among RUBiS$_2$-Client and RUBiS$_2$-Apache, RUBiS$_2$-Tomcat, RUBiS$_2$-Sql are all close to 1.0, while the *inter-cluster correlation coefficients* between RUBiS$_2$-Client and other VMs are less than 0.2. We have evaluated the correlation values for window sizes of 30, 60, and 90. These results demonstrate that the correlation detection mechanism based on the three metrics of CPU utilization, PacketsIn and PacketsOut can correctly identify VM groups that communicate with each other.

Net-Cohort is also able to differentiate the same classes of instances that run different tasks. This is because different tasks typically show different resource usage patterns which can then be used by Net-Cohort to identify the *intra-cluster correlations* within the ensemble and the *inter-cluster correlations* outside of the ensemble. For example, Figure 9 (a) and (b) show the CPU usage patterns of two Hadoop instances respectively. The first Hadoop instance reads text files and counts how often words occur. The second Hadoop instance computes exact binary digits of the mathematical constant π. From Figure 9 (a) and (b), we learn that the "CPU spikes" and "CPU valleys" of the slaves that belongs to the same ensemble consistently occur together, thereby creating the correlations needed by Net-Cohort.

For intuition about the ensemble identification process, we also look at how the correlation-based hierarchical tree is built by Net-Cohort.

Figure 10 shows the hierarchical trees constructed using the decreasing level of dependency strength. We first determine a $N \times N$ correlation strength matrix, and then run the hierarchical clustering algorithm over the distance matrix for the cases without packet sniffing. The hierarchical trees generated by **R** [6], which shows the calculated dependent links between the VMs being observed. This demonstrates that Net-Cohort can effectively expose most of the underlying dependencies between the VMs within our testbed.

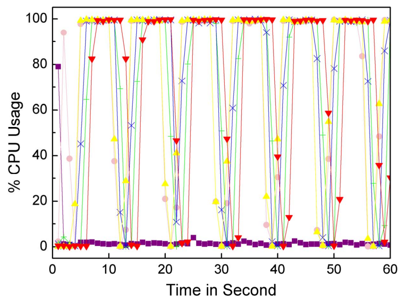


Figure 9 (a). CPU pattern of Hadoop instance of word

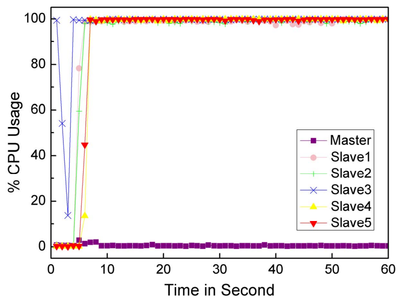


Figure 9 (b). CPU pattern of Hadoop instance of computing PI.

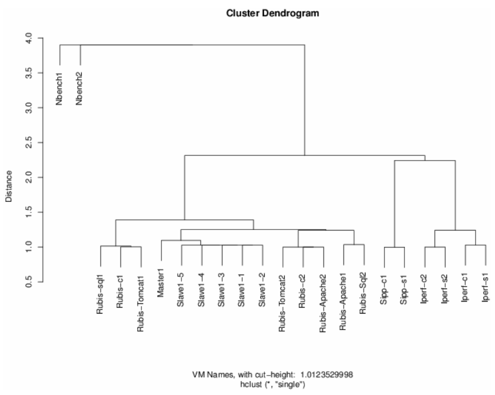


Figure 10. Hierarchical tree generated by R with window size = 90.

6.2 Net-Cohort Benefit Evaluation

After determining the VM ensembles and hierarchical clustering of VM groups, we next evaluate the benefit of these inputs for better VM placement on hosts. In doing so, we first look into the impact of available bi-section bandwidth on applications. To study this impact, we set the crossover bandwidth among RUBiS components from 1Mbps to 100Mbps to reveal how bandwidth changes influence RUBiS performance.

Figure 11 (a) reports the throughput in requests per second as a function of available bandwidth among components. Bandwidth is varied from 1Mbps to 100Mbps. Every RUBiS workload includes three stages: up ramp, runtime session, and down ramp. This shows that the throughput of all three stages of RUBiS are quite sensitive to the available bandwidth, since the throughput corresponding to 100Mbps bandwidth is more than that of the 1Mbps available network bandwidth by about 20 times. Figure 11 (b) reports the response time in millisecond per request as the available bandwidth among components. Bandwidth is varied from 1Mbps to 100Mbps. It shows that as bandwidth increases, the average response time of all three stages of RUBiS is significantly reduced by over 1000 times.

To test VM placement, we deploy the RUBiS components (Apache server, Tomcat server, database server and client), SIPp components (SIPp server, SIPp client) to domains with 100Mbps limited bi-section bandwidth. The incoming and outgoing bandwidth is capped by Linux control groups and traffic shaping tools at VMM's layer. To aggravate competition for bi-section bandwidth, we simultaneously run the Iperf instance, the mapping of VMs to hosts and racks is shown in Figure 12. We apply Net-Cohort black-box methods to determine the VM ensembles, use that input to come up with the moves through VM placement engine, and remap VMs onto hosts. We then compare application performance results for initial and final placements. Here, we assume that CPU and memory resources are not a bottleneck.

Figure 13(a) shows that RUBiS throughput increases from 27 request/second, 20 request/second, and 19 request/second to 68 request/second, 97 request/second, and 41 request/second for up ramp, runtime session, and down ramp, respectively. Figure 13(b) shows that RUBiS response time per request decreases from 6787 millisecond, 15245 millisecond, and 28037 millisecond to 468 millisecond, 31 millisecond, and 25 millisecond for up ramp, runtime session, and down ramp, respectively.

Figure 14 shows that compared to the initial placement, the throughput of the first Hadoop instance increases from 735.5 Kbyte/second to 1103.3 Kbyte/second, and the throughput of the second Hadoop instance increases from 1180.5 Kbyte/second to 1609.9 byte/second.

Figure 15 shows that SIPp's quality of service in terms of number of failed calls is improved significantly. Calls failed due to the co-located VMs saturate the limited bandwidth. The average number of failed calls decreases from 250.62 to 19.64.

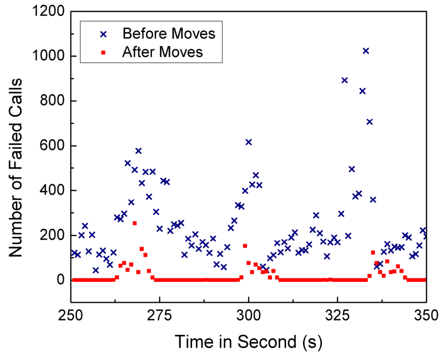


Figure 15. Comparison of SIPp performance in number of failed calls for initial vs final VM placements.

Experimental results clearly indicate the benefits of bandwidth-aware VM placement, thereby demonstrating that information obtained using Net-Cohort can help improve placement and migration actions while reducing utilization of network resources in virtualized data centers. These potential improvements are motivation for future work in which we are integrating Net-cohort with existing management utilities. (e.g., see vManage [19]).

6.3 Packet Sniffer Overheads

The first phase of correlation detection has no extra overhead because basic VM level statistics are already reported by management systems. The second phase of distance identification has a complexity of O(N^2) where N equals to number of VMs, costing several seconds. The third phase of clustering has a complexity of O(N^2) costing several seconds. Packet sniffing is turned on a very selective set of hosts and VMs.

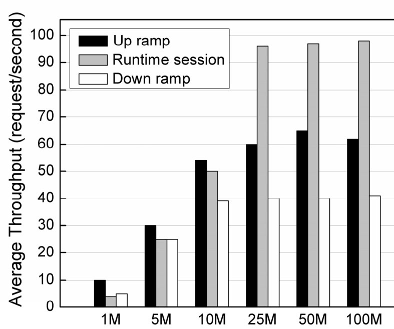


Figure 11 (a). RUBiS throughput changes as a function of available bandwidth.

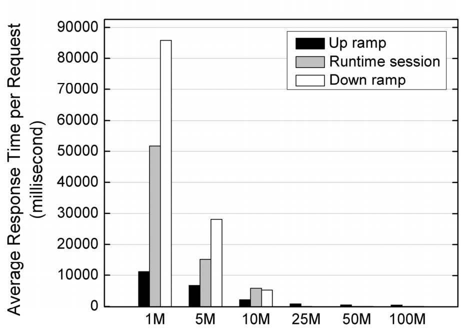


Figure 11 (b). RUBiS response time changes as a function of available bandwidth.

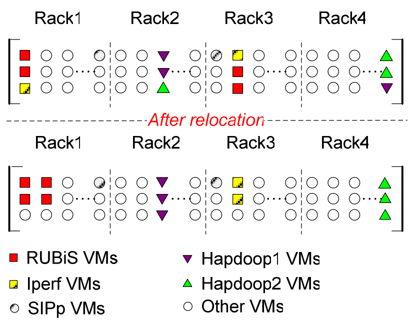


Figure 12. Before and after VM/PM mappings.

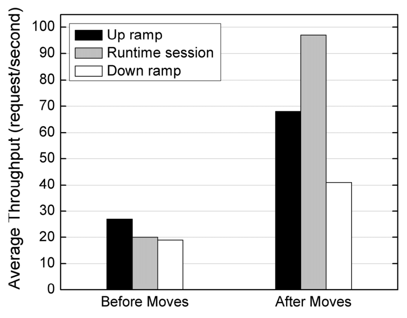


Figure 13 (a). Comparison of RUBiS throughput for initial vs. final VM placements.

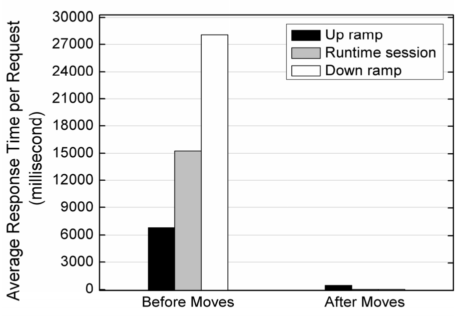


Figure 13 (b). Comparison of RUBiS response time for initial vs. final VM placements.

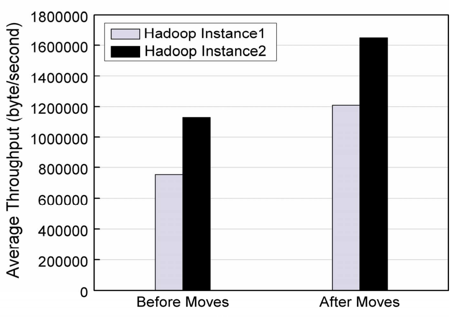


Figure 14. Comparison of Hadoop throughput for initial vs. final VM placements.

We empirically estimate the CPU and memory overheads due to the packet sniffer running in Domain0. Figure 16 shows the additional CPU usage when using the packet sniffer for 5VMs and 10VMs as a function of time. Note that the CPU overhead can be negligible if the transferred traffic is not heavy and the number of VMs monitored is small. However, the packet sniffing technique may trigger irregular fluctuations in CPU usage when capturing large amount of packets, switching among interfaces or periodically flushing I/O buffers of logs to disks. Effects would worsen for larger numbers of VMs on a host or when moving from 1Gbps to 10 Gbps NICs. Latency per packet would also be higher if the packet sniffer is being run in a separate VM due to multiple context switches.

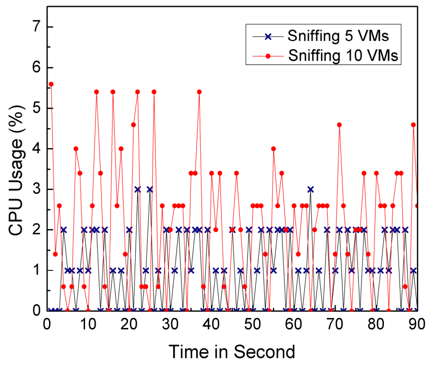


Figure 16. CPU overhead of packet sniffer when sniffing different number of VMs.

In summary, since the packet sniffer is not transparent to users in that its use may introduce heavy CPU overheads, we apply it only when the black-box methods require additional information to accurately cluster VMs.

7. CONCLUSIONS

Net-Cohort presents a set of lightweight, non-intrusive techniques for identifying communicating VMs so as to enable VM placements and migrations that reduce the pressure on bi-section bandwidth in consolidated data centers.

Net-Cohort benefits data center administrators in several ways. In general, it informs them about the VM ensembles currently being run in the data center, thereby enabling them to manage their systems for entire applications rather than individual VMs. More specifically, it makes it possible to place communicating VMs in ways that preserve scarce bi-section bandwidth, which can improve application performance and reduce effects on other bandwidth-sensitive services. Net-cohort's simple design can be realized with available statistics and in a scalable fashion, without requiring changes to data center facilities. It has three unique characteristics:

- It does not require knowledge about application semantics, the implementation of the platform, or a priori information about communication paths.
- There is no need to modify applications, middleware, workloads, or platforms.
- It has extremely low impact on system performance.

Net-Cohort's methods are fully implemented, but additional work is required for using it to continuously monitor and manage data center systems at scale [25]. This includes practical steps like the efficient correction for time series correlation calculation due to potential lack of synchronization among VMs' clocks, and the efficient runtime construction and deployment of monitoring and analysis 'overlays' with data center systems and applications. Moreover, we are working on a pre-clustering module that is capable of filtering out background traffic (i.e., management traffic), since such traffic might otherwise be interpreted as intra-ensemble communications. Further, it would be interesting to understand how methods like Net-Cohort can be integrated with management solutions like VMware's DRS and Hyper-V's PRO. Finally, we note that future 'flat' data center networks, like those developed by companies like Brocade, HP, and Juniper Networks, may shift the utility of Net-Cohort from providing a means to reducing the use of network bi-section bandwidth to instead, serving as a management tool that enables administrators to recognize and manipulate entire applications or properties rather than individual VMs.

8. REFERENCES

[1] Aurora. http://www.zurich.ibm.com/aurora/.

[2] Cisco Data Center Infrastructure 2.5 Design Guide. www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf.

[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] Microsoft's Performance and Resource Optimization (PRO). http://technet.microsoft.com/en-us/library/cc764283.aspx.

[6] R Project. http://www.r-project.org/.

[7] VMware Distributed Resource Scheduler (DRS). http://www.vmware.com/products/drs/.

[8] S. Agarwala, F. Alegre, K. Schwan, J. Mehalingham, "*E2EProf: Automated End-to-End Performance Management for Enterprise Systems*", In *the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN'07), June 2007.

[9] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen*, "Performance Debugging for Distributed Systems of Black Boxes*," In *Proceedings of SOSP*, 2003.

[10] R. Apte, L. Hu, K. Schwan, A. Ghosh, "*Look Who's Talking: Discovering Dependencies between Virtual Machines Using CPU Utilization*," In *the 2nd USENIX Workshop on Hot Topics in Cloud Computing* (HotCloud' 10), June 2010.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. "*Hedera: Dynamic Flow Scheduling for Data Center Networks,*" In *USENIX NSDI*, April 2010.

[12] S. Bagchi, G. Kar, and J. L. Hellerstein, "*Dependency analysis in distributed systems using fault injection: Application to problem determination in an e-commerce environment*," In *Proc. 12th Intl. Workshop on Distributed Systems: Operations & Management*, Nancy, France, Oct. 2001.

[13] A. Brown, G. Kar, and A. Keller, "*An active approach to characterizing dynamic dependencies for problem determination in a distributed environment*," In *Proc. 7th IFIP/IEEE Intl. Symp. on Integrated Network Management*, Seattle,WA, May 2001.

[14] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "*Pinpoint: Problem determination in large, dynamic systems*," In *Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, pages 595–604, Washington, DC, June 2002.

[15] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "*Automating network application dependency discovery: Experiences, limitations, and new solutions*," In *OSDI*, San Diego, California, Dec. 2008.

[16] L. Golab, D. DeHaan, E. D. Demaine, "Identifying. Frequent Items in Sliding Window over On-line Packet Streams," Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC '03), Florida, USA.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, "*VL2: A Scalable and Flexible Data Center Network*," ACM SIGCOMM 2009, August 2009.

[18] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. "*Performance and Availability Aware Regeneration for Cloud Based Multitier Applications,*" In *the 40th IEEE/IFIP International Conference on Dependable Systems and Network* (DSN 2010) *Performance and Dependability Symposium*, Illinois, Chicago, June 2010.

[19] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, K. Schwan: "*vManage: loosely coupled platform and virtualization management in data centers*," ICAC 2009, 127-136.

[20] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J.C. Mogul, "*SPAIN: COTS Data center Ethernet for Multipathing over Arbitrary Topologies*," In *Proc. NSDI*, 2010, pp.265-280.

[21] S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "*Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers*", In *Proceedings of the ACM SIGCOMM Conference*, New Delhi, India, August 2010.

[22] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat, "*Pip: Detecting the unexpected in distributed systems*," In *Proc. NSDI*, San Jose, CA, May 2006.

[23] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, "*WAP5: black-box performance debugging for widearea systems*," In *WWW*, 2006.

[24] B.C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang, "*vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities*," In *USENIX ATC*, 2009.

[25] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, M. Wolf, "A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-Scale Data Centers," In *ICAC*, 2011.

[26] Y. Zhang, A. Su and G. Jiang, "*Understanding Data Center Network Architectures in Virtualized Environments: A View from Multi-Tier Applications,*" Elsevier Computer Networks, Vol. 55, No. 9, 2011.