2012 32nd IEEE International Conference on Distributed Computing Systems

v-Bundle: Flexible Group Resource Offerings in Clouds

Liting Hu[1], Kyung Dong Ryu[2], Dilma Da Silva[2], Karsten Schwan[1]

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

[2]IBM T.J.Watson Research Center
Yorktown Heights NY, USA
{kryu, dilmasilva}@us.ibm.com

***Abstract*—Traditional Infrastructure-as-a-Service offerings provide customers with large numbers of fixed-size virtual machine (VM) instances with resource allocations that are designed to meet application demands. With application demands varying over time, cloud providers gain efficiencies through resource consolidation and over-commitment. For cloud customers, however, this leads to inefficient use of the cloud resources they have purchased. To address cloud customers' dynamic application requirements, we present a new cloud resource offering, called v-Bundle, which makes flexible the exchange of resource capacity among multiple VM instances belonging to the same customer. Specifically targeting network resources, for each customer application, we first use DHT-based techniques to achieve an initial VM placement that minimizes its use of the datacenter network's bi-section bandwidth. When VMs' networking requirements change, the customer can then use v-Bundle to trade the networking resources allocated to her application. v-Bundle maintains information about network resources with any-cast tree-based methods implemented as extensions of the Pastry pub-sub core. Experimental evaluations show that the approach can scale well to thousands of hosts and VMs, and that v-Bundle can provide customers with better bandwidth utilization and improved application quality of service through borrowing extra bandwidth when needed, at no additional cost in terms of the total resources allocated to the customer.**

Keywords-cloud computing; virtualization; v-bundle; bi-section bandwidth; pastry overlay.

I. INTRODUCTION

Recently, there has been a dramatic increase in the popularity of Infrastructure-as-a-Service (IaaS) clouds, including Amazon EC2 [1], Rackspace [6], Eucalyptus [3], Nebula [4], and Openstack [5]. Such systems provide compute resources on-demand, bill on a pay-as-you-go basis, and allow multiple customers to share a common pool of virtualized resources. Also provided is a variety of pre-configured, templated images containing various operating systems and pre-installed software. The purchase of cloud resources proceeds as follows. (1) The customer estimates her application needs, using VM configuration parameters like the desired CPU and memory capacities and outbound network bandwidth; also specified are the operating systems and the numbers of required VMs (interchangeably called instances). (2) The customer finalizes the contracts with the cloud provider for some period of use, e.g., a one-year term. (1) and (2) fully define the resource package purchased by the customer.

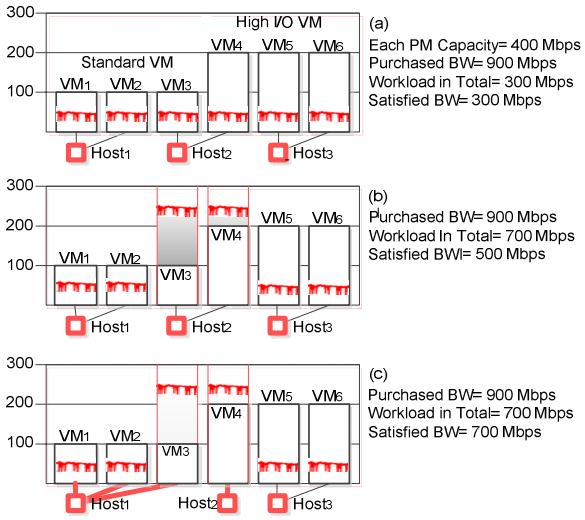

Figure 1. Example of traditional resource provisioning & v-Bundle's provisioning.

IaaS providers, in turn, allocate virtual resources according to the customers' settings, and such resources are billed regardless of hosted applications' actual current needs. Therefore, when a customer's applications experience dynamic variations lasting for longer periods of time, this may lead to inefficient or even unfair use of her cloud resources: the customer may pay for computing capacity she does not actually receive.

v-Bundle makes use of the dynamically changing requirements of multiple applications within the group of VMs purchased by a single customer. With v-Bundle's resource trading service, each customer can independently optimize the use of her cloud resources. A simple example illustrates the v-Bundle approach and potential benefits. At the top of Figure 1, we depict a customer's resource package with 6 VMs (instances) placed over 3 host physical machines (PMs) each with 2 VMs. The customer bought 3 standard VMs with 100 bandwidth Mbps capacity and 3 high I/O VMs with 200 Mbps bandwidth capacity, to match the typical requirements of her front-end vs. back-end services realized by these VMs. Suppose that each physical host only has a 400 Mbps network interface. When all application workloads are as light as 50 Mbps, all instances' demands can be satisfied (see (a)). However, when workloads exceed per-VM capacity allocations (see VM_3 and VM_4 in (b)), the de-facto standard cloud resource offering provides only a max of 500 Mbps out of the total 900 Mbps total bandwidth available to the customer. As

1063-6927/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDCS.2012.61

shown in (c), v-Bundle-based discovery of available resources makes it possible to live migrate VMs to relocate over-loaded VMs to borrow unused CPU cycles, memory or bandwidth from lightly loaded ones, as long as all of those VMs belong to the same customer. In this case, VM_3 is migrated to a lightly loaded server PM_1 and borrows compute capacity from VM_1 and VM_2, thereby relieving the resource shortage of PM_2.

While it is apparent how the v-Bundle approach can be applied to CPU and memory resources, the networking resource presents unique challenges to the realization of the v-Bundle concept. This is because of currently prevalent hierarchical networks in datacenter systems, with switch infrastructures (starting with top of rack (ToR) switches) that are increasingly over-provisioned at layers above ToR (see Section 2). Those unique challenges are as follows:

1) Bi-section Bandwidth: is a critical, scarce, and expensive resource. Recent studies [11] [16] [18] have shown that servers in different racks have to share the up-links from top of rack switches (i.e., ToRs) that are typically 1:5 to 1:20 oversubscribed. IaaS providers, however, are unaware of the hosted instances' communication patterns. Thus, they use simple methods, like random or CPU-usage-based placement, to map VM instances onto servers, as long as there are sufficient resources left. For customers, this may result in the undesirable outcome that they receive less resources than what they paid for.

2) Decentralized Management: today's IaaS offerings can span thousands of servers with VMs from thousands of customers. This presents challenges to centralized methods for resource management. For example, the time complexity for the load balancing step is O(#VMs×#hosts), plus the time required for cost-benefit analysis. For a cluster containing 100 hosts and 10000 VMs, for instance, it takes nearly 10 minutes for a load-balancing algorithm to run [9]. For a cloud provider that needs to manage thousands of customers, it will be too costly to set up such a manager for each customer.

v-Bundle is a novel multi-tenant virtualized datacenter resource scheduler. It has the following key attributes:

- *Bi-section bandwidth preserving*: When booting up VMs, it leverages a peer-to-peer protocol, Pastry [16], to implement a topology-aware algorithm to place "chatting VMs" geographically close to each other. We assume that VMs from the same customer are more likely to communicate with each other than with others.
- *Scalability:* For the second step, rebalancing VMs, v-Bundle leverages a multicast tree protocol, Scribe, to implement a decentralized resource discovery algorithm to shuffle workload, at VM granularity, between over-loaded and less-loaded servers.

The Java Scribe/Pastry-based implementation of v-Bundle is evaluated on a virtualized infrastructure consisting of 15 hosts and 225 VMs. These VMs run a mix of bandwidth-intensive workloads. Experimental results show that v-Bundle can scale well, optimally utilize the available bi-section bandwidth, and improve the applications' QoS by taking advantage of workload variations belonging to the same customer.

The remainder of this paper is organized as follows. Section 2 describes v-Bundle's placement algorithm. Section 3 describes v-Bundle's resource shuffling algorithm. Sections 4 and 5 evaluate v-Bundle with simulated experiments and real experiments, respectively. Section 6 discusses related work. We conclude with directions for future work in Section 7.

II. V-BUNDLE' TOPOLOGY-AWARE PLACEMENT ALGORITHM

Focusing on network resources, v-Bundle proposes a novel placement algorithm that uses a DHT-based routing protocol to guide initial VM placement. Using this network-efficient method is a prerequisite for optimizing customers' use of network resources.

As illustrated in Figure 2, if wrongly placing an ensemble of frequently communicating "chatting" VMs across multiple racks, this can negatively affect the services provided by the application [15]. First, as the shared up-links from ToRs become saturated, intra-ensemble communications may be delayed, and 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. Therefore, if v-Bundle wants to optimize the network resource usage for each customer, it must start by first making its best efforts to avoid allocating or relocating "chatting" VMs across multiple racks.

A. Pastry's Peer-to-Peer Overlay

In Pastry [16], each participating node is assigned an identifier (*nodeId*) that is used to identify nodes and route messages. Given a message and a destination *nodeId*, Pastry routes the message to the node whose *nodeId* is numerically closest to the destination within limited O(logN) hops.

1) Routing table: Each Pastry node has two routing structures, a routing table and a leaf set. The routing table consists of node characteristics (IP address, latency information, and Pastry ID) organized in rows by the length

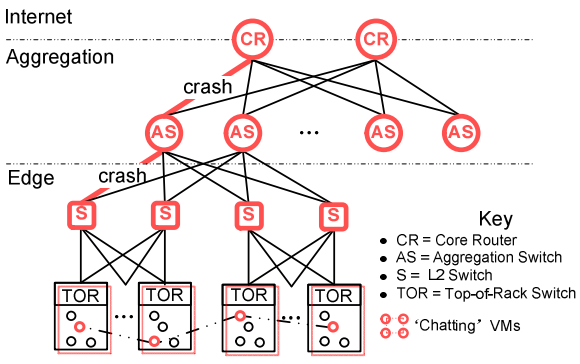

Figure 2. Example of shared up-links from TORs crash, causing performance degradation for many VMs.

of common prefix. When routing a message, each node forwards it to the node in the routing table with the longest prefix in common with the destination *nodeId*.

2) Leaf set: The leaf set for a node is a fixed number of nodes that have the numerically closest *nodeId* to that node. This assists nodes in the last step of routing messages and in rebuilding routing tables when nodes fail.

B. "Topology-aware" Placement Algorithm

Assume there are *N* servers in the cloud. As shown in Figure 3, first, a centralized certificate authority assigns each server with a unique Id from 2, 2^{128}-1. Note that the *nodeIds* are assigned to reflect the physical proximity. Specifically, *nodeIds* are assigned branches to branches to be in accordance with the hierarchical structure of the data center. The numerically adjacent nodes are also physically close to each other. Second, when a new customer arrives, say IBM, the customer name will be hashed to be a key (e.g., *hash*(IBM)) and tagged to all of that customer's VMs. Third, when preparing to boot up a new VM instance, a message is encapsulated with that new VM's attributes, resulting in a query to the destination ID (which equals to *hash*(IBM)). The server whose *nodeId* is numerically close to the key would receive the query and check whether to boot up the new VM. If the target server fails to satisfy the request, the query will be forwarded to its neighbor set of servers. The neighbor set M contains the *nodeIds* and IP addresses of |M| nodes that are closest (according the proximity metric) to the local node. The process repeats until the query is satisfied.

C. Benefits and Design Rationale

Having described v-Bundle's topology-aware algorithm, we now explain its benefits and rationale.

1) Reducing the bi-section bandwidth bottleneck: The algorithm is based on the assumption that VMs from the same customer have a higher potential to communicate with each other than with others. Since such "chatting VMs" share the same key, they are more likely to be placed geographically close within the same server or rack. As a result, their inter-VM traffic is less likely to traverse bottleneck switches or routers. When new VMs arrive, their placement follows the same rule.

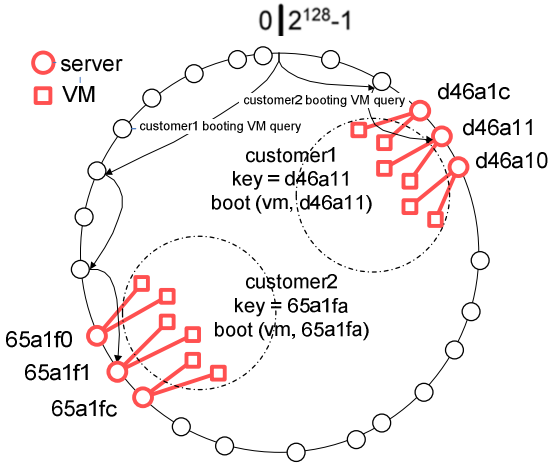

Figure 3. v-Bundle's topology-aware VM Placement.

2) Simple and decentralized solution: v-Bundle uses a query-response model to alleviate the load on front end servers performing management. There is no need to ask them to periodically acquire a snapshot of the cloud, to check free resources or calculate the least loaded target servers. Instead, the new VMs' boot-up query can be processed in parallel and in a distributed fashion.

3) Flexible abstraction: v-Bundle offers cloud customers a flexible abstraction to express their expectation when placing VMs. For example, if a customer wishes to place VM group 1 and VM group 2 close to each other, she can simply ask the cloud provider to tag the two groups with the same key.

III. v-Bundle's Decentralized Resource Shuffling Algorithm

For the second step of cloud resource offering, resource shuffling between VMs, v-Bundle proposes a novel search and exchange algorithm, using Pastry's any-cast facility to guide VM redistribution. The goal is to permit each customer to leverage the workload variations experienced by her multiple and different applications. Resource requirements are stated in terms of the minimal and maximal resources for each VM instance, and VM migration is used to deal with "cold" (lightly loaded) vs. "hot" (over- or heavily loaded) servers. Specifically, resource rebalancing under workload variation is enabled by creating VM groups whose members advertise some spare resource, e.g., bandwidth, and then give it away in response to the receipt of any-cast messages from servers that seeks to consume the bandwidth. Members leave the group when they no longer have extra bandwidth available or when their utilization exceed some threshold value (e.g., above group average).

A. Background

1) Scribe's Group Management

Scribe is an application-level group communication system built upon Pastry. Scribe manages groups and maintains a spanning tree containing the members of each group. All nodes join Pastry, and subsequently, nodes may join and leave groups. Scribe can handle group sizes varying from one to millions, and it efficiently supports rapid changes in group membership [12][22]. It uses a pseudo-random Pastry key to name a group, called *groupId*. Usually, the *groudId* is the hash of the group's textual name concatenated with its creator's name. To create a group, a Scribe node asks Pastry to route a CREATE message using the *groupId* as the key. The node responsible for that key becomes the root of the group's tree.

Scribe leverages the existing proximity-aware Pastry overlay. To join a group, a node routes a JOIN message towards the *groupId*. The message will continue to be routed till it reaches a node in that tree. The route traversed by the message to the group multicast tree would be added. As a result, Scribe can efficiently support large numbers of

groups, arbitrary numbers of group members, and groups with highly dynamic membership.

2) Multicast and Anycast

Scribe supports two important properties: multicast and any-cast. Multicast can be used to develop a hierarchical aggregation tree: a fundamental abstraction for scalability. Any-cast can be used to perform distributed resource discovery.

- Multicast: Any node in the overlay can create a group; other nodes can join the group and then multicast message to all members of the group. Multicast messages are disseminated from the rendezvous point along the multicast tree.
- Any-cast: this is implemented using a distributed depth-first search of the group tree. Any node in the overlay can any-cast to a Scribe group by routing the message towards the *groupId*. Pastry's local route convergence ensures that the message reaches a group member near the message's sender with high probability.

B. VM's Attributes: Reservation and Limit

Amazon EC2 sets a static attribute tuple for each VM instance, e.g., a High-CPU instance with configuration $<1.7GB\ Mem, 5\ EC2\ Compute\ Units, 1GB\ Bandwidth>$. The cloud provider guarantees that the specified CPU, memory, and bandwidth are always available for the instance. However, VMs are not allowed to use more resources than the specified value. As shown in this paper, as a result, the customer might waste idle resources she actually bought.

Therefore, unlike Amazon EC2, v-Bundle's VMs specify *reservations* and *limits* for CPU, memory, or bandwidth resources, to express their demands for the cloud. *Reservation* specifies a minimal guaranteed amount of a resource. The VM is only allowed to power on if the reservation is available and that amount of CPU, memory, or bandwidth can be guaranteed even when the server is over-loaded. *Limit* specifies an upper bound for CPU, memory, or bandwidth for a VM instance. That is, more resources can be allocated to a VM than the reservation amount if the application workload changes. However, the allocated resources will never exceed the specified limit.

C. Decentralized Resource Management

The workflow of v-Bundle's resource rebalancing is as follows. (1) Each server self-identifies its status as load shedder or load receiver (here, we use the term "load" and "utilization" interchangeably). The baseline is the average utilization of corresponding resource plus a threshold. (2) After self-identification, the load shedder initiates a query including the hosted over-loaded VMs' information to find *k* nearby receivers.

To clarify, we next describe sample scenarios. Assume there are 42 VM instances in one customer's resource package, hosted over 7 servers, in which 3 of them are bandwidth over-loaded (assume bandwidth is the only bottleneck resource and each VM instance consumes 10% server's bandwidth).

The first step would be to calculate the average bandwidth utilization. As shown in Figure 4, we create two aggregation trees, rooted at two rendezvous points, to disseminate messages containing the aggregated information. These two trees are the *BW_Capacity* tree and the *BW_Demand* tree. All servers subscribe to them. The server with *nodeId* numerically closest to the *topicId* acts as the rendezvous point for the associated multicast tree. For example, if *hash(BW_Demand)* equals to 1100, the node with the same identifier or closest identifier like 1101 or 1099 will be the root of *BW_Demand* tree.

Each server has local data stored as a set of (*attributeName*, *value*), such as (*BW_Capacity*, 10). Periodically, the leaf node updates its local state/value and passes the update to its parent, and then each successive enclosing subtree updates its aggregate value and passes the new value to its parent. The system then computes the desired aggregate value at each layer up the tree until the root holds the desired value. Finally, the root sends the result down the tree to all members.

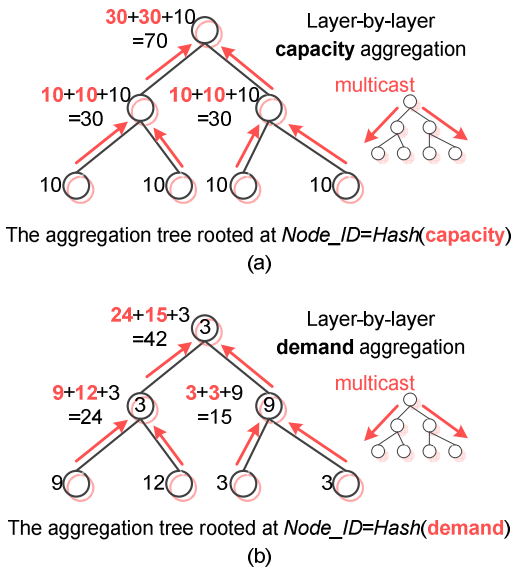

Figure 4. v-Bundle's aggregation trees.

After each server has obtained the cluster level information, including the current bandwidth demand and the total bandwidth capacity through publishing, it can calculate the average utilization at its own end, and can identify itself as shedder or receiver. In our example, the average utilization line would be 42/70=60%, as shown in Figure 5.

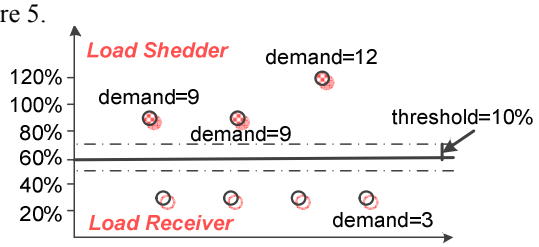

Figure 5. Self-identification as load shedder or load receiver.

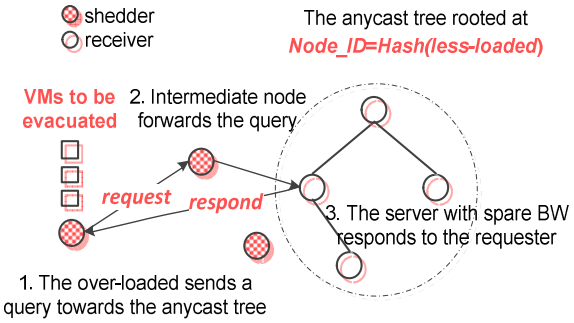

Figure 6. Self-identification as load shedder or load receiver.

The second step would be reliable resource rebalancing. As shown in Figure 6, to do so, it will subscribe to a new any-cast tree, called the *Less-Loaded* tree. Suppose a load shedder wants to evacuate its VMs to loader receivers. Let's now look at how workload exchange happens:

- Step1: The load shedder periodically sends a load-balance query towards the *less-loaded* tree. The query contains its evacuated VMs' information like the bandwidth requirement.
- Step2: The query is encapsulated as an any-cast message and forwarded to an any-cast address with *groupId* equal to *hash(less-loaded)*, which means that at least one of the servers can accept messages for that any-cast address. v-Bundle prefers topologically closest candidates among the target candidates to receive the query first so as to maintain the bisectional bandwidth preserving placement.
- Step 3: The first load receiver that receives the query will check (1) if it has sufficient reserved bandwidth to accept the new VM; (2) after accepting the new VM, if the server's updated bandwidth utilization is still under the cluster mean plus a threshold, which avoids possible oscillation for back-and-forth shedding/receiving. If both checks pass, this server sends a response back and hold part of its bandwidth waiting for the new VM, else it forward the query to other members.
- Step 4: If the load shedder receives an acknowledgement from the load receiver indicating acceptance, the shedder initiates that VM's migration. Meanwhile, the load shedder updates its bandwidth utilization periodically; it stops sending load-balance query if its bandwidth utilization drops down the average line.

D. Implementation Highlights

The implementation includes two parts: (i) a hypervisor-based controller that shapes the resource usage of each running instance; (ii) a cross-hypervisor interface that define a new aggregation abstraction, which paves the way for utilizing the DHT system's internal trees for aggregation and for achieving scalability.

Focusing on network bandwidth, v-Bundle uses control groups combined with Linux traffic shaping (TC) to control the volume of traffic being sent into a network by each VM in a specified period. v-Bundle uses TC to set rate and ceil. Rate means the guaranteed bandwidth available for a given VM and ceil is short for ceiling, which indicates the maximum bandwidth that VM is allowed to consume.

The cross-hypervisor interface aims at providing each participating server with a summary view of global information. We define a new aggregation abstraction, which is across all servers in the system. Each server can show interest in one or more topics and subscribe to one or more aggregation trees correspondingly. Each physical server has local data stored as a set of (*topic*, *attributeName*, *value*) tuples such as (*configuration*, *numCPUs, 16*). We associate an aggregation function with each topic. For each *level-i* subtree, the root of the subtree calculates the aggregated value of its children's data and sends updates to its parent.

Each node has one or more topic managers that keep track of the topics in which it is interested. Each topic manager maintains the linkages to its ancestor and descendants. We refer to a store of (*ChildNodehandle*, *value*) tuples as an information base. Each intermediate node contains its children's reduction information base. These values and parent-children structures changes as new node updates, joins and leaves. Each node periodically retrieves the children's updated reduction information bases and the root node publishes the result.

E. Benefits and Design Rationale

First, v-Bundle enables flexible resource shuffling between VM instances by setting thresholds. For example, if the hosted application is a VoIP-like bandwidth aggressive instance, the threshold should be small in order to provide timely relief to "hot" servers.

Second, the load shedder can exploit Scribe's any-cast facility to discover and manage free bandwidth at small time scales. The any-cast tree is self-organizing and self-repairing, and any-cast completes after visiting a small number of nodes (O(log(n)), where n is the number of servers in the datacenter. This means that the cost for discovering any load receiver is limited to O(log(n)) hops.

Third, similar to the placement algorithm, the resource rebalancing algorithm is also simple and decentralized, avoiding the need for a central manager that could become a performance bottleneck and single point of failure.

IV. Simulated Experiment Evaluation

Our experimental setup is limited to 15 servers. These servers are 15 dual-core dual-socket, each with two Intel Xeon 5150 processors, 16GB of memory, and 80GB hard drives. They 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 and the oversubscription ratio is set to be 8:1.

The limited size of this setup prompts us to create a larger-scale simulation in order to evaluate v-Bundle's DHT decentralized management. Specifically, using one JVM to represent one node, we emulate up to H=3000 servers and V=5000~10000 VMs for 5 customers.

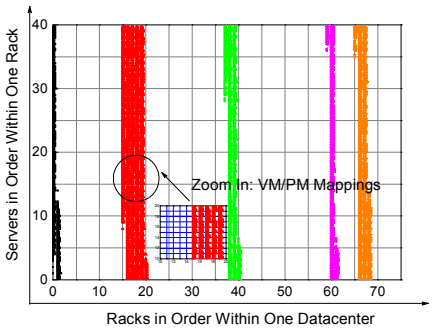

Figure 7. VM/PM mappings when instanciating 5000 VMs on top of 3000 servers for 5 customers using v-Bundle's approach:
a, black dot: customer<Accolade>'s VM;
b, red dot: customer<Beenox>'s VM;
c, green dot: customer<Crystal>'s VM;
d, pink dot: customer<Deck13>'s VM;
e, orange dot: customer<Epyx>'s VM.

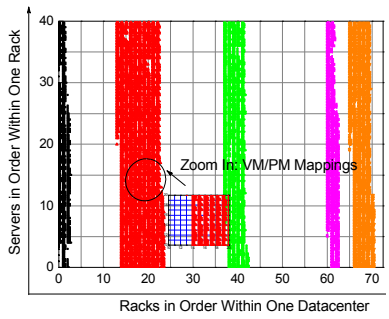

Figure 8. (a) v-Bundle: After 5000 VMs, another 5000 new VMs are instanciated on the same 3000 servers for the same 5 customers using v-Bundle's approach:
a, black dot: customer<Accolade>'s VM;
b, red dot: customer<Beenox>'s VM;
c, green dot: customer<Crystal>'s VM;
d, pink dot: customer<Deck13>'s VM;
e, orange dot: customer<Epyx>'s VM.

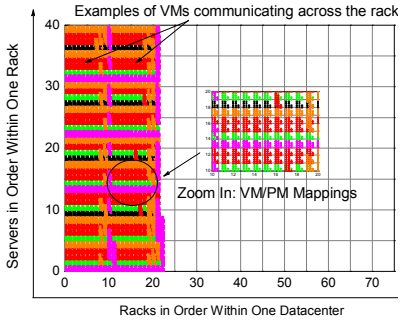

Figure 8. (b) Greedy: After 5000 VMs, another 5000 new VMs are instanciated on the same 3000 servers for the same 5 customers using greedy-based's approach:
a, black dot: customer<Accolade>'s VM;
b, red dot: customer<Beenox>'s VM;
c, green dot: customer<Crystal>'s VM;
d, pink dot: customer<Deck13>'s VM;
e, orange dot: customer<Epyx>'s VM.

A. DHT-based Placement Evaluation

Figure 7 shows the snapshot of v-Bundle's datacenter in terms of virtual to physical machine mappings, where the X-axis represents all racks in the datacenter, the Y-axis represents the servers in order within one rack, and the crossing point is a VM's position. The adjacent servers across racks will be assigned remote *nodeIds* so as to avoid VMs belonging to the same customer to happen to be placed on them. The graph clearly demonstrates that the VMs belonging to the same customer are placed geographically close to each other. The VMs belonging to different customers are dispersed evenly across the whole data center. Therefore, the inter-VM traffic traversing the bottleneck switch or router is minimized.

Another 5000 new VMs are provisioned for each customer. Figure 8(a) shows the updated snapshot of v-Bundle in the datacenter. It is shown that although the number of VMs is doubled for each customer, VMs sharing the same key are still placed together to the greatest extent within the same rack or server. The reason lies in that keys are chosen randomly and mapped to geographically diverse servers, so peers who are adjacent in keys have space to grow or shrink.

For comparison, a greedy placement's snapshot is displayed in Figure 8(b). The greedy algorithm makes decisions on the basis of information at hand without considering the effects these decisions may have in the future. It places the new coming VMs on the first server it finds with enough resources. As a result, the initial snapshot looks like Figure 8(b), in which newcomer VMs fail to find a place adjacent to the VMs with which they collaborate and have to traverse long paths to communicate with each other.

B. Decentralized Resource Rebalancing Evaluation

To evaluate v-Bundle's resource rebalancing, we create a scenario in which most of the servers hosting VMs for a given customer show load imbalance. The reason might be that some of the customer VMs' resource demands achieve their peak while some other VMs have low demand. When using v-Bundle, we expect to see that the hot servers are relieved, ending with a more balanced snapshot.

The threshold is the margin added to the average utilization line. If a server's utilization is greater than the average line by a certain margin, it self-identifies as a load shedder. Otherwise, if it is smaller than the average line by a certain margin, it self-identifies as a load receiver. The balancing happens as a result of servers making VM exchanges among each other in order to take advantage of workload variations and thus maximize the utilization of the resources purchased by the customer. The goal of load balancing is to ensure that all servers are within range 0 to *mean+threshold*. Once satisfied, resource rebalancing will stop. Of course, if the average line is really low, the resource rebalancing will not be triggered. The threshold value and tolerance level may be set up jointly by data center administrators and customers.

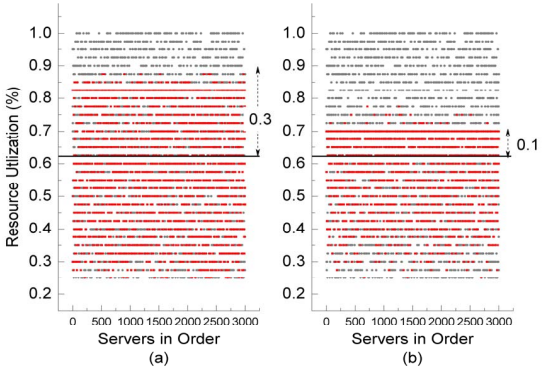

Figure 9. Initial snapshot and after snapshot of 3000 servers (75000 VMs) with different resource utilizations: a, gray dot: servers' status before rebalancing; b, red dot: servers' status after rebalancing.

Figure 9 shows the bandwidth utilization's snapshot of 3000 servers before and after the v-Bundle rebalancing process, where the X-axis represents servers in order, and the Y-axis represents the resource utilization in percentage terms. The average utilization line is 0.6226. From Figure 9, we can see that before v-Bundle rebalancing takes effect, about half of the servers are overloaded. When the threshold equals 0.3, the servers over 90% utlization experience relief (see Figure 9(a)). When the threshold equals 0.1, the servers over 70% experience relief (see Figure 9(b)). This demonstrates that the smaller the threshold, the more servers may be involved in rebalancing, resulting in more exchanges among over-loaded and less-loaded servers.

To demonstrate that v-Bundle can make rebalancing decisions quickly with increasing numbers of servers, Figure 10 shows the instance rebalancing process for 30 servers and 3000 servers, where the X-axis represents the time in minutes, and the Y-axis represents the standard deviation (SD) of all servers' utilizations. To simplify the result, we ignore that migration itself consumes bandwidth. Between about 32 minutes and 60 minutes, two sharp decreases in utilization are observed, separated by a rebalancing interval of 25 minutes. We can see that, when setting the same default threshold (0.183), 3000 servers and 30 servers use similar time to reach stable snapshots. The reason lies in that shedding load requests are initiated spontaneously by each individual server and VMs' exchanges happen in parallel within the resource bundle. The decision is made locally and thus, the time cost does not increase linearly or exponentially with the total number of servers.

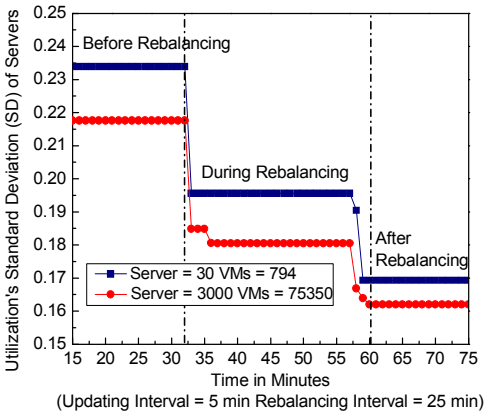

Figure 10. Instance rebalancing process for 3000 servers (75350 VMs) and 30 servers (794 VMs).

Figure 11 shows performance gains due to v-Bundle's instance rebalancing, where the X-axis represents the time in minutes, and the Y-axis represents the sum of bandwidth resource in Mbps. It is found that between 0~33 minutes, there is an obvious difference between the resource demand in total and the actual satisfied resource in total. Because some VMs' demands reach the peak value but are bounded by the hardware limits of the underlying servers, other VMs' demands decrease to some low value and end up

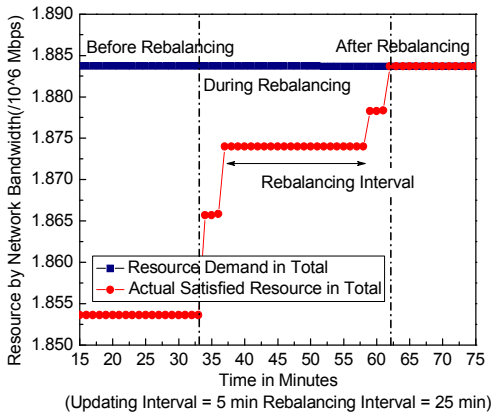

Figure 11. Resource gains during Instance rebalancing for 3000 servers (75350 VMs)

wasting the underlying server's resources, Between 34~61 minutes, v-Bundle takes effect. v-Bundle initiates 2 rounds of load shedding at about minutes 33 and 57. We can see that the actual satisfied resource in total is approaching the resource demand in total due to v-Bundle moving unfairly treated VMs to free servers, and thus taking advantage of customer-level workload variations. After 62 minutes, all VMs' demands are satisfied. It is only at this time that the customer paying for some level of QoS actually receives it.

V. Real Experiment Evaluation

We have asserted that v-Bundle can help improve the QoS of applications belonging to one customer, by taking advantage of workload variations. In this section, we evaluate v-Bundle on a real virtualized environment consisting of 15 servers and 225~300 VMs.

We instantiate 255~300 virtual machines in total, each of which is configured to use 128MB of RAM. Xen 3.1.2 is the virtual machine monitor on each host for all experiments, and the host kernel for XenLinux is a modified version of Linux 2.6.18.

A. Workload and Metrics

A mix of bandwidth-aggressive applications is chosen to create the bandwidth bottleneck and unbalanced scenario:

SIPp [7] is a traffic generator for the SIP protocol. It can establish real client and server sessions and initiate/release thousands of calls with a given rate. It can also send media (RTP) traffic through RTP echo and RTP/pcap replay. Media can be audio and video. Call rate (calls per seconds) starts from 800, increases by 10 every second, with the maximum rate set to 3000 and total calls to 1000K. The following statistics are gathered to measure SIPp's execution performance: the ratio of failed call and application's response time partition.

SiPp is the application requiring QoS and we want to provide it with "efficient" use of purchased group resource, and in order to generate resource contention, we run Iperf [8]. Iperf is a commonly used network workload generation tool that can create TCP and UDP data streams and measure the throughput of the network carrying them. Iperf includes

client and server functionality, and can measure the throughput between the two ends, either unidirectionally or bi-directionally. We continuously run Iperf pairs to generate interference traffic and thus introduce the bandwidth bottleneck.

B. *Experiment Methodology and Results*

After evaluating the decentralized management of v-Bundle using simulation, we next evaluate the benefit of v-Bundle for optimizing resource usage and thereby improving the QoS of applications. In the first step, a mix of bandwidth aggressive applications is booted unevenly on the hosts and run with an increasing workload. After a while, when the bandwidth becomes the major bottleneck and the unbalancing scenario is detected, v-Bundle will create the less-loaded anycast tree and start the VM/PM rebalancing process. Cost-benefit analysis is applied before any actual migrations are performed.

Migration may be either live or cold, with the distinction based on whether the instance is running at the time of migration. In live migration, the instance continues to run during its transfer, whereas with cold migration, the VM is paused, saved, and sent to another physical server. Migration is possible only if administrators choose appropriate storage solutions (e.g., SAN, NAS, etc.) to ensure that guest OS file systems are also available on their destination servers. In our example, we use live migration and export a shared storage for guest domains via NFS.

Figure 12 shows SIPp's performance gains in terms of number of failed calls due to v-Bundle's instance rebalancing, where the X-axis represents time in seconds, and the Y-axis represents the number of failed calls. It is found that before the 300th second, since Iperf VM and SIPp VM are co-located on the same server, when their demands achieve peak value, they are bounded by hardware limits. As a bandwidth sensitive application, SIPp experiences performance loss in terms of numbers of failed calls. Between the 300th second and the 375th second, v-Bundle takes effect by initiating VMs relocations. After the 375th second, the SIPp application's quality of service is improved greatly.

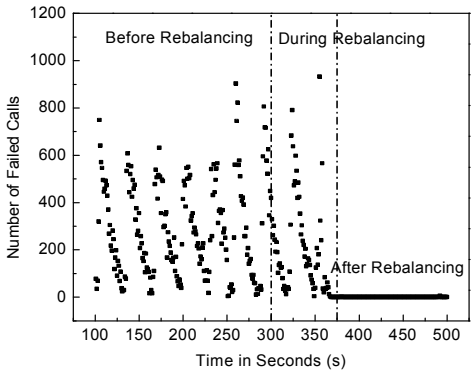

Figure 12. Number of failed calls of SIPp application before rebalancing versus after reblancing.

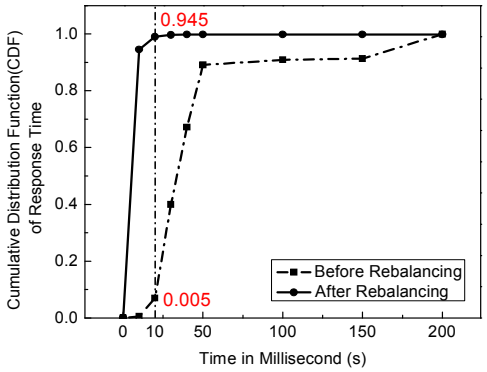

Figure 13. Cumulative distribution function of response time of SIPp application before rebalancing versus after reblancing.

Figure 13 shows the SIPp's performance gains in terms of the cumulative distribution of response time due to v-Bundle's instance rebalancing. Before rebalancing, only 10% of the calls have a response time of less than 10 milliseconds. After rebalancing, the response time of 90% of the calls is less than 10 milliseconds.

C. *Overhead Analysis*

As undue numbers of tree reconfigurations and migrations will cause unexpected overheads, it is important to make sure that the overall QoS cost is not exceeding the overall QoS benefit.

We analytically estimate the computation time for all pub-sub operations: subscriptions, unsubscriptions, and publications. We also measure the aggregation time for a message that is sent by a subscriber till it is received by the root/publisher, and the per host communication overheads in terms of messages/round. All times are measured using the nanoTime method in J2SE 1.6.0 and averaged over 1000 measurements on 3 Dell PowerEdge 1950 compute servers containing Intel Xeon 5150 processors using Java 1.6.0. Table 1 summarizes the overhead for v-Bundle operations.

TABLE I. COMPUTATION OVERHEAD FOR V-BUNDLE OPERATIONS

	Subscribe(ms)	Unsubscribe(ms)	Publish(ms)	Dataset(bytes)
Publisher	-	-	1.383	128
Subscriber	$0.775\times\|T\|+0.0211$	$0.229\times\|T\|$		128

Figure 14 shows the aggregation latency of v-Bundle by tracking a message from the time it is sent by each subscriber till the message is aggregated to the publisher. Note that the message is sent periodically by each subscriber with an interval of 5 minutes, so we show two lines in which the blue line represents the time cost without adding an updating interval, and the red line calculates the total time cost for pushing the message from leaves to root.

Observe that the latency increases linearly as the number of nodes increases exponentially. The reason lies in that, as the number of nodes increases, the height of the aggregation/dissemination tree also increases. For example, the height of the tree is bound to 3 layers for 256 nodes and

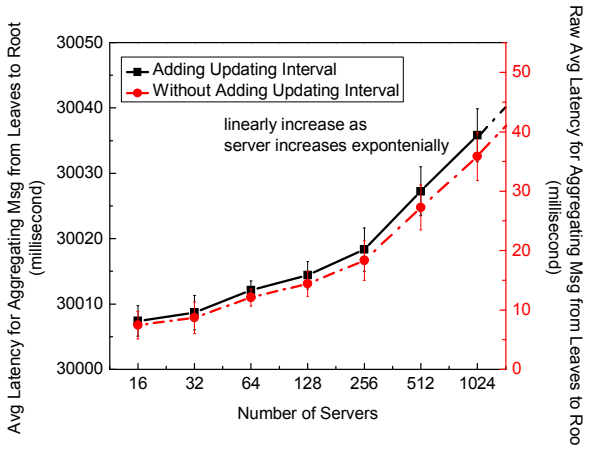

Figure 14. Latency of aggregating info from leaves to root versus the number of nodes for v-Bundle.

increases to 3 layers for 1024 nodes. An increase in height by one incurs an additional latency of 10ms (local-area network latency), thereby increasing the overall latency. However, within the overall latency, the time added at every node due to v-Bundle is quite small (about 1-2 ms).

We also measure overhead per host in terms of messages/round. We instantiate a FreePastry ring with different numbers of servers and compute the number of messages sent for each server. We break the overall overhead into two categories: overall communication needed for maintaining aggregation framework and for running v-Bundle on top. Figure 15 shows the cumulative distribution function (CDF) of the total overhead for 512 and 1024 servers. Note that for 90% of the servers, the overall overhead is less than 40 KB/round and 140 msg/round for the 1024 host setup. Furthermore, the overhead grows organically, in a very logarithmic fashion, and even for 100K nodes, it should only go to 80 KB/round based on the projection.

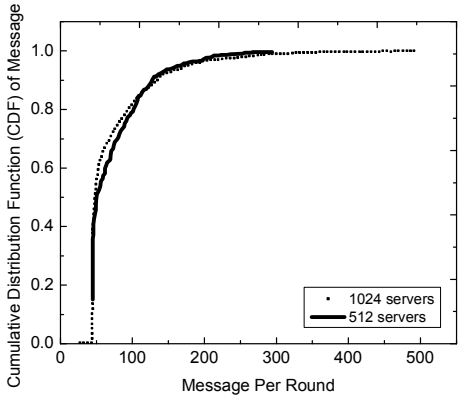

Figure 15. Cumulative distribution function of message per round for hosts.

VI. Related Work

To the best of our knowledge, we are the first to provide a cloud-level service for customers to efficiently use their purchased cloud resources combination. Further, by avoiding performance bottlenecks due to limited bi-section bandwidth, we offer a path to cloud providers for transitioning from current hierarchical to future 'flat' datacenter networks. In this section, we compare and contrast the approach presented in this paper with related work in the literature.

A. Cloud Resource Offering

Based on Xen virtualization technology for Linux and XenServer for Windows, Rackspace Cloud [6] and Amazon EC2 [1] support customers to easily boot up an instance an instance automatically, and scalable image repositories make it possible to configure and boot up 1 to 50 VMs in minutes. While the providers can consolidate customer VMs, no support is given to customers to better manage the cloud resources they have purchased. If a customer buys 10 instances with 100GB bandwidth total, to run some application, then, even when 9 of those instances only use 1GB bandwidth during demand lulls, the customer cannot shed the redundant instances in favor of her 10th instance, forcing her to pay for the wasted 90GB bandwidth while one instance may still starve.

B. Dedicated Physical Severs Offering

Given a private data center consisting of dedicated physical servers, VMware DRS [9] continuously monitors utilization across all servers in the resource poll, and intelligently allocates available resources among VMs. Unlike Amazon EC2 or Rackspace, VMs are not tied to the booting server on which they are started. When a VM experiences increased load due to its hosted applications, VMware DRS will work with VMware VMotion [10] and use live migration to redistribute VMs and re-balance workload across physical servers. A central manager is used to monitor each server's utilization and track each VM's resource demand. This is feasible for small private clouds, but fails to serve large public clouds running on hundreds or thousands of servers. In addition, it does not support to "exchange" resource deficit and surplus across VMs of the same customer.

C. Techniques to improve hosted applications' QoS

Q-Clouds [17] tunes resource allocations to mitigate performance interference effects. It transparently provisions additional resources when deploying VMs. The additional resources ("head room") can be used for compensating for performance interference between VMs, the goal being to prevent or ameliorate the effects of consolidation and sharing seen by applications. As there is little control over how tenants share the network, to avoid starved VMs, Seawall [20] uses an edge-based rate controller to achieve max-min fairness across tenant VMs by sending traffic through congestion-controlled hypervisor-to-hypervisor tunnels. The rate controller takes as input the packets received and sent by the compute node and congestion

feedback from the network and recipient. Similar to Seawall, SoftUDC [14] uses hypervisor rate limiters to control the network utilization of different tenants within a shared datacenter.

The difference between the above techniques and v-Bundle lies in that v-Bundle rebalances the VMs via live migrations among "hot" servers and "cold" servers so as to borrow the unused resources from the customer's own instances. In contrast, Q-Cloud asks for additional resources ("head room") from the cloud provider to make better QoS guarantees to resource starved applications. Seawall and SoftUDC choose to hold back or set a limit on those aggressive applications and thus leave space for those that suffer.

VII. Conclusions and Future work

Focusing on the network bandwidth resource, v-Bundle presents a set of light-weight, non-intrusive and decentralized methods for optimizing the resource usage for each customer by enabling computing capacity switching among her VMs, improving their applications' quality of service, meanwhile helping the cloud provider save the critical bi-section network bandwidth.

v-Bundle benefits cloud customers and cloud providers in several ways. In general, it provides the customers a new offering, called virtual resource bundle, in which their instances are no longer "fixed size" and always being allocated with full resources even when they do not need them. Instead, VMs belonging to the same customer can shuffle their compute capacity using VM migration. At the same time, the cloud provider obtains a proactive way to optimize the usage of critical bi-section datacenter network bandwidth. The VMs that are more likely to communicate are placed geographically close to each other by using DHT-based routing protocols, thus preserving scarce bi-section bandwidth and reducing potential ill effects on other bandwidth-sensitive services. v-Bundle's simple design can be realized in a scalable fashion, without requiring changes to datacenter facilities.

v-Bundle's methods are fully implemented, but additional work is required for using it to continuously monitor and manage data center systems at scale [21]. This includes improving the decentralized resource shuffling algorithm by considering multiple metrics like CPU, memory, and bandwidth. Moreover, we are working on a cost-benefit module that is capable of predicting the overhead due to live migrations and the benefit from resource shuffling.

References

[1] http://aws.amazon.com/ec2/

[2] http://www.appengine.google.com/

[3] http://www.eucalyptus.com/

[4] http://nebula.com/

[5] http://www.openstack.org/

[6] http://www.rackspace.com/managed_hosting/private_cloud/

[7] http://sipp.sourceforge.net/

[8] http://iperf.sourceforge.net/

[9] http://www.vmware.com/products/drs/overview.html

[10] http://www.vmware.com/products/vmotion/overview.html

[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] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "*An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays*," Infocom 2003, San Francisco, CA, April, 2003.

[13] 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.

[14] M. Kallahalla, M. Uysal, R. Swaminathan, D. E. Lowell, M. Wray, T. Christian, N. Edwards, C. I. Dalton, and F. Gittler, "*SoftUDC: A Soware-Based Data Center for Utility Computing*," Computer, 37(11):38–46, 2004.

[15] G. Lee, N. Tolia, P. Ranganathan, R. H. Katz, "Topology-aware resource allocation for data-intensive workloads," In Proceedings of ApSys'2010, pp.1~6.

[16] 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.

[17] R. Nathuji, A. Kansal, "*Q-Clouds : Managing Performance Interference Effects for QoS-Aware Clouds*," in Proceedings of the 5th European conference on Computer systems, (ACM, 2010), p. 237–250.

[18] 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.

[19] A. Rowstron and P. Druschel, "*Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*," IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001.

[20] A. Shieh, S. Kandula, A. Greenberg, C. Kim, "*Seawall: performance isolation for cloud datacenter networks*," In the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud' 10), June 2010.

[21] 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.

[22] P. Yalagandula, M. Dahlin, "*A Scalable Distributed Information Management System*," Sigcomm'04, Aug. 30-Sept. 3, 2004, Portland, Oregon, USA.