Towards Predictable Datacenter Networks

Hitesh Ballani, Paolo Costa, Thomas Karagiannis, Ant Rowstron
Microsoft Research, Cambridge, UK

Technical Report
MSR-TR-2011-72

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1. INTRODUCTION

The simplicity of the interface between cloud providers and tenants has significantly contributed to the increasing popularity of cloud datacenters offering on-demand use of computing resources. Tenants simply ask for the amount of compute and storage resources they require, and are charged on a pay-as-you-go basis.

While attractive and simple, this interface misses a critical resource, namely, the (intra-cloud) network.[1] Cloud providers do not offer guaranteed network resources to tenants. Instead, a tenant's compute instances (virtual machines or, in short, VMs) communicate over the network shared amongst all tenants. Consequently, the bandwidth achieved by traffic between a tenant's VMs depends on a variety of factors outside the tenant's control, such as the network load and placement of the tenant's VMs, and is further exacerbated by the oversubscribed nature of datacenter network topologies [1]. Unavoidably, this leads to high variability in the performance offered by the cloud network to a tenant [2–5] which, in turn, has several negative consequences for both tenants and providers.

–*Unpredictable application performance and tenant cost.* Variable network performance is one of the leading causes for unpredictable application performance in the cloud [3], which is a key hindrance to cloud adoption [6,7]. This applies to applications across the spectrum; from user-facing web applications [3,8] to transaction-processing web applications [9] and even MapReduce-like data-intensive applications [3,10]. Further, since tenants pay based on the time they occupy their VMs, and this time is influenced by the network, tenants implicitly end up paying for the network traffic; yet, such communication is supposedly free (hidden cost).

–*Limited cloud applicability.* The lack of guaranteed network performance severely impedes the ability of the cloud to support various classes of applications that rely on predictable performance. The poor and variable performance of HPC and scientific computing applications in the cloud is well documented [11,12]. The same applies to data-parallel applications like MapReduce that rely on the network to ship large amounts of data at high rates [10]. As a matter of fact, Amazon's ClusterCompute [13] addresses this very concern by giving tenants, at a high cost, a dedicated 10 Gbps network with no oversubscription.

–*Inefficiencies in production datacenters and revenue loss.* The arguments above apply to not just cloud datacenters, but to any datacenter with multiple tenants (product groups), applications (search, ads, MapReduce), and services (BigTable, HDFS, GFS). For instance, in production datacenters running MapReduce jobs, variable network performance leads to poorly performing job schedules, and severely impacts datacenter throughput [10,14,15]. Also, such network-induced application unpredictability makes job scheduling qualitatively harder and hampers programmer productivity, not to mention significant loss in revenue [15].

These limitations result from the mismatch between the desired and achieved network performance by tenants which hurts both tenants and providers. Motivated by these factors, this paper tackles the challenge of extending the interface between providers and tenants to explicitly account for network resources, while maintaining its simplicity. Our overarching goal is to allow tenants to express their network requirements while ensuring providers can flexibly account for them. To this end, we propose "virtual networks" as a means of exposing tenant requirements to providers. Tenants, apart from getting compute instances, are also offered a virtual network connecting their instances. The virtual network isolates tenant performance from the underlying infrastructure. Such decoupling benefits providers too– they can modify their physical topology without impacting tenants.

The notion of a virtual network opens up an important question: *What should a virtual network topology look like?* On one hand, the abstractions offered to tenants must suit application requirements. On the other, the abstraction governs the amount of multiplexing on the underlying physical network infrastructure and hence, the number of concurrent tenants. Guided by this, we propose two novel abstractions that cater to application requirements while keeping tenant costs low and provider revenues attractive. The first, termed *virtual cluster*, emulates a "virtual Ethernet cluster" suited for data-intensive applications like MapReduce. The second, named *virtual oversubscribed cluster*, offers an oversubscribed tree-like structure that suits applications that feature local communication patterns.

Hence, *the primary contribution of this paper is the design of virtual network abstractions that explore the trade-off between the guarantees offered to tenants, the tenant cost and provider revenue.* We further present Oktopus, a system that implements our abstractions. Oktopus maps tenant virtual networks to the physical network in an online setting, and enforces these mappings. Using extensive simulations and deployment on a 25-node testbed, we show that expressing requirements through virtual networks enables a symbiotic relationship between tenants and providers; tenants achieve better and predictable performance while the improved datacenter throughput (25-435%, depending on the abstraction) increases provider revenue.

A key takeaway from Oktopus is that our abstractions can be deployed today: they do not necessitate any changes to tenant applications, nor do they require

[1] In the rest of this paper, we use "network" to refer to the intra-cloud network connecting compute instances within a datacenter. The "external network" is referred to as such.

Study	Source	Provider	Duration
A	[5]	Amazon EC2	NA
B	[3]	Amazon EC2	31 days
C/D/E	[2]	3 providers	1 day
F/G	[17]	Amazon EC2	1 day
H	[4]	Amazon EC2	1 day

Table 1: Studies measuring intra-cloud network bandwidth.

changes to routers and switches. Further, offering guaranteed network bandwidth to tenants opens the door for explicit bandwidth charging. Using today's cloud pricing data, we find that virtual networks can reduce median tenant costs by up to 74% while ensuring revenue neutrality for the provider.

On a more general note, we argue that predictable network performance is a small yet important step towards the broader goal of offering an explicit cost-versus-performance trade-off to tenants in multi-tenant datacenters [16] and hence, removing an important hurdle to cloud adoption.

2. NETWORK PERFORMANCE VARIABILITY

Network performance for tenants in shared datacenters depends on many factors beyond the tenant's control– the volume and kind of competing traffic (TCP/UDP), placement of tenant VMs, etc. Here, we discuss the extent of network performance variability in cloud and production datacenters. Overall, we find that there is significant variability that cannot be ignored - jobs may take up to 4-5 times their median completion time.

2.1 Cloud datacenters

A slew of recent measurement studies characterize the CPU, disk and network performance offered by cloud vendors, comment on the observed variability, and its impact on application performance [2–5,17]. We contacted the authors of these studies and summarize their measurements of the intra-cloud network bandwidth, i.e., the TCP throughput achieved by transfers between VMs in the same cloud datacenter. Table 1 presents relevant information about the measurements, while Figure 1 plots the percentiles for the network bandwidth. The figure shows that tenant bandwidth can vary significantly, *even by a factor of five or more in some studies* (A, B, F and H).

While more work is needed to determine the root-cause for such bandwidth variations, anecdotal evidence suggests that the variability is correlated with system load (EU datacenters, being lightly loaded, offer better performance than US datacenters) [3,14], and VM placement (e.g., VMs in the same availability zone perform better than ones in different zones) [3]. Further, as mentioned in Section 1, such network performance variability leads to poor and unpredictable application performance [3,8–10].

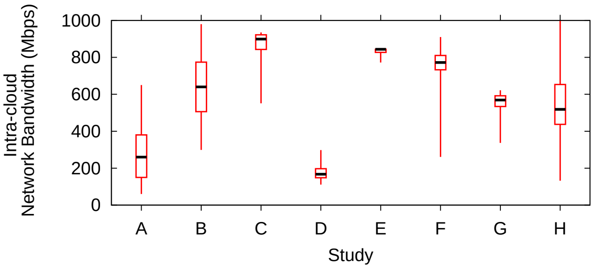
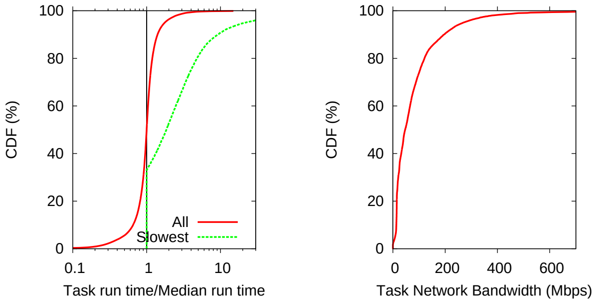


Figure 1: Percentiles (1-25-50-75-99th) for intra-cloud network bandwidth observed by past studies.



(a) Variability in runtime.

(b) Variability in bandwidth

Figure 2: Production datacenter running data analytics jobs.

2.2 Production datacenters

Production datacenters are often shared amongst multiple tenants, different (possibly competing) groups, services and applications, and these can suffer from performance variation. We characterize such variation based on data from [15] that reflects a production datacenter with tens of thousands of commodity servers. These servers run data analytics jobs for a major search provider using a data-parallel framework similar to MapReduce [18], Dryad [19] and Cosmos [20]. Individual jobs are comprised of phases (map, reduce, join, etc.) while phases contain multiple tasks, each operating on a different part of the input for the phase.

Since tasks for a given phase perform the same operation, their runtime is expected to be similar. To verify this, we look at the runtime of tasks in each phase. The CDF for the ratio of the runtime of individual tasks to the median task runtime is plotted in Figure 2(a). While many tasks finish close to the median task, the distribution has a long tail. This is important since, in most cases, the slowest task dictates the phase finish time and hence, the job finish time. The figure also shows the CDF for the ratio of the runtime of the slowest task to the median task. In 25% of cases, the runtime of the slowest task is more than 4 times the median task (1.85 times in 50% of the cases)! These results show that task runtimes vary significantly, and this can adversely impact job completion times [10].

Variability in task runtimes can result from a number of factors– unequal distribution of data amongst tasks, failures, network performance, etc. We perform an analysis similar to [15] and determine the fraction of slowest tasks (with runtime greater than 1.5 times the median) that could be explained by the amount of data read across cross-rack links where congestion typically occurs. We find that 20% of such slow tasks can be explained through the amount of cross-rack transfers. Since the logs do not contain information about data read by tasks from machines in the same rack, we were unable to extend this analysis to account for all data read from the network. However, the result shows that *at least* 20% of the slowest tasks can be attributed to the network. Further, the bandwidth achieved by tasks that read data across cross-rack links can vary by more than an order of magnitude (from 8.4 Mbps at the 5^{th} percentile to 270 Mbps at the 95^{th} percentile, Figure 2(b)).

In summary, we observe significant variability in network performance which negatively impacts application performance. Further, our evaluation shows that in both cloud and production datacenters, the mismatch between required and achieved network performance hurts datacenter throughput and hence, provider revenue. Since our proposed abstractions cover both cloud and production datacenters, we will henceforth use the term "multi-tenant" to refer to both.

3. VIRTUAL NETWORK ABSTRACTIONS

In multi-tenant datacenters, tenants request virtual machines (VMs) with varying amounts of CPU, memory and storage resources. For ease of exposition, we abstract away details of the non-network resources and characterize each tenant request as $<N>$, the number of VMs requested. The fact that tenants do not expose their network requirements hurts both tenants and providers. This motivates the need to extend the tenant-provider interface to explicitly account for the network. Further, the interface should isolate tenants from the underlying network infrastructure and hence, prevent provider lock-in. Such decoupling benefits the provider too; it can completely alter its infrastructure or physical topology, with tenant requests being unaffected and unaware of such a change. To this end, we propose virtual networks as a means of exposing tenant network requirements to the provider. Apart from specifying the type and number of VMs, tenants also specify the virtual network connecting them.

The "virtual" nature of the network implies that the provider has a lot of freedom in terms of the topology of this network, and can offer different options to tenants for different costs. Beyond the overarching goal of maintaining the simplicity of the interface between tenants and providers, the topologies or *virtual network abstractions* should be guided by two design goals:

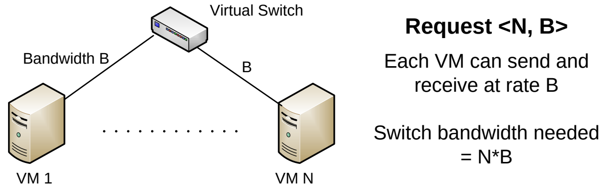

Virtual Switch
Bandwidth B
B
VM 1
VM N
Request <N, B>
Each VM can send and receive at rate B
Switch bandwidth needed = N*B

Figure 3: Virtual Cluster abstraction

1. *Tenant suitability.* The abstractions should allow tenants to reason in an intuitive way about the network performance of their applications when running atop the virtual network.

2. *Provider flexibility.* Providers should be able to multiplex many virtual networks on their physical network. The greater the amount of sharing possible, the lesser the tenant costs.

Motivated by the above discussion, we propose two novel abstractions for virtual networks in the following sections. Our designs aim at balancing these competing goals through the richness of the virtual network topology.

3.1 Virtual Cluster

The "Virtual Cluster" abstraction is motivated by the observation that in an enterprise (or any private setting), tenants typically run their applications on dedicated clusters with compute nodes connected through Ethernet switches. This abstraction, shown in figure 3, aims to offer tenants with a similar setup. With a *virtual cluster*, a tenant request $<N, B>$ provides the following topology: each tenant machine is connected to a virtual switch by a bidirectional link of capacity B, resulting in a one-level tree topology. The virtual switch has a bandwidth of $N * B$. This ensures that the virtual network has no oversubscription and the maximum rate at which the tenant VMs can exchange data is $N * B$. However, this data rate is only feasible if the communication matrix for the tenant application ensures that each VM sends and receives at rate B. Alternatively, if all N tenant VMs were to send data to a single destination VM, the data rate achieved will be limited to B.

Since a *virtual cluster* offers tenants a network with no oversubscription, it is suitable for data-intensive applications like MapReduce, BLAST, etc. For precisely such applications, Amazon's Cluster Compute provides tenants with compute instances connected through a dedicated 10 Gbps network with no oversubscription.

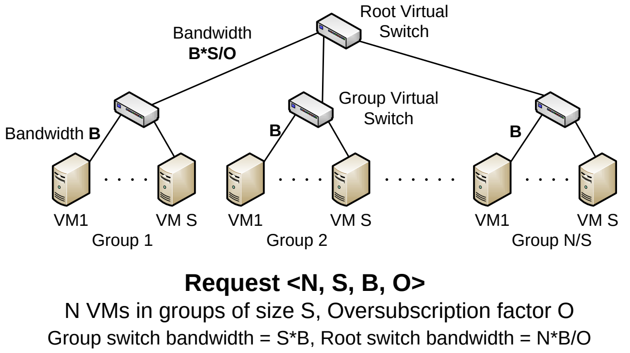


Figure 4: Virtual Oversubscribed Cluster abstraction

This may be regarded as a specific realization of the *virtual cluster* abstraction with $<N, 10\ Gbps>$.

While a network with no oversubscription is imperative for data-intensive applications, this does not hold for many other applications [21,22]. Instead, a lot of cloud bound applications are structured in the form of components with more intra-component communication than inter-component communication [23,24]. A "Virtual Oversubscribed Cluster" is better suited for such cases; it capitalizes on application structure to reduce the bandwidth needed from the underlying physical infrastructure compared to virtual clusters, thereby improving provider flexibility and reducing tenant costs. We describe this abstraction in the following section.

3.2 Virtual Oversubscribed Cluster

With a *virtual oversubscribed cluster*, a tenant request $<N, B, S, O>$ entails the topology shown in Figure 4. Tenant machines are arranged in groups of size S, resulting in $P = \frac{N}{S}$ groups[2]. VMs in a group are connected by bidirectional links of capacity B to a (virtual) group switch. The group switches are further connected using a link of capacity $B' = \frac{S*B}{O}$ to a (virtual) root switch. The resulting topology has no oversubscription for intra-group communication through the group switches. However, inter-group communication has an oversubscription factor O, i.e., the aggregate bandwidth at the VMs is O times greater than the bandwidth at the root switch. Hence, this abstraction closely follows the structure of typical oversubscribed datacenter networks. Note, however, that O *neither depends upon, nor requires physical topology oversubscription.*

The maximum data rate with this topology is still $N * B$. Yet, the localized nature of the tenant's bandwidth demands resulting from this abstraction allows the provider to fit more tenants on the physical network. Compared to *virtual cluster*, this abstraction does not offer as dense a connectivity but, as our evaluation shows, has the potential to significantly limit tenant costs. Hence, in effect, by incentivizing tenants to expose the flexibility of their communication demands, the abstraction achieves better multiplexing which benefits both tenants and providers. Amazon's EC2 Spot Instances [25] is a good example of how tenants are willing to be flexible, especially when it suits their application demands, if it means lowered costs.

Other topologies. A number of network abstractions have been proposed in other contexts and could potentially be offered to tenants today. For example, many topologies have been studied for HPC platforms, such as multi-dimensional cubes, hypercube and its variants, and even more complex topologies such as Butterfly networks, de Bruijn, and ShuffleNet [26]. Similarly, SecondNet [27] provides tenants with bandwidth guarantees for pairs of VMs, resulting in a clique virtual topology.

We believe that these existing proposals suffer from several drawbacks due to their dense connectivity, and hence significantly limit the true flexibility a multi-tenant datacenter can provide. First, they are of interest to a small niche set of applications (violating goal 1). Second, their dense connectivity also makes it difficult for the provider to multiplex multiple tenants on the underlying network infrastructure (violating goal 2). For instance, the analysis in [27] shows that with the oversubscribed physical networks prevalent in today's datacenters, only a few tenants demanding clique virtual networks are sufficient to saturate the physical network. This hurts the provider revenue and translates to high tenant costs. Finally, these abstractions offer multiple links per VM which makes it difficult to account for resource bottlenecks; for instance, the disk bandwidth at a VM may prevent the VM from saturating the N links offered by the clique topology, resulting in resource wastage.

Table 2 illustrates how the topologies discussed compare with respect to our design goals. The *virtual cluster* provides rich connectivity to tenant applications that is independent of their communication pattern but limits provider flexibility. The *virtual oversubscribed cluster* utilizes information about application communication patterns to improve provider flexibility. The clique abstraction, chosen as a representative of existing proposals offers very rich connectivity but severely limits provider flexibility and reflects only a few applications.

4. Oktopus

To illustrate the feasibility of virtual networks, we present Oktopus, a system that implements our abstractions[3]. The provider maintains a datacenter containing

[2] While we focus on groups of uniform size, the *virtual oversubscribed cluster* abstraction and the algorithm presented in Section 4.2 apply to variable-sized groups as well.

[3] Oktopus provides predictable performance, and is named

Abstraction	Max Rate	Suitable for applications	Provider Flexibility	Tenant Cost
Virtual Cluster	O(N)	All	Medium	Medium
Oversub. cluster	O(N)	(Almost) All	High	Low
Clique	O(N^2)	Limited	Very Low	Very High

Table 2: Virtual network abstractions present a trade-off between application suitability and provider flexibility.

physical machines with slots where tenant VMs can be placed. With Oktopus, tenants requesting VMs can opt for a (virtual) cluster or a (virtual) oversubscribed cluster to connect their VMs. Further, to allow for incremental deployment, we also support tenants who do not want a virtual network, and are satisfied with the status quo where they simply get some share of the network resources. Two main components are used to achieve this:

- *Management plane.* A logically centralized network manager (*NM*), upon receiving a tenant request, performs admission control and maps the request to physical machines. This process is the same as today's setup except that the NM needs to further account for network resources and maintain bandwidth reservations across the physical network.

- *Data plane.* Oktopus uses rate-limiting at endhost hypervisors to enforce the bandwidth available at each VM. This ensures that no explicit bandwidth reservations at datacenter switches are required.

The network manager implements allocation algorithms to allocate slots on physical machines to tenant requests in an online fashion. For tenant requests involving a virtual network, the NM needs to ensure that the corresponding bandwidth demands can be met while maximizing the number of concurrent tenants. To achieve this, the NM maintains the following information– (i). The datacenter network topology, (ii). The residual bandwidth for each link in the network, (iii). The empty slots on each physical machine, and (iv). The allocation information for existing tenants, including the physical machines they are allocated to, the network routes between these machines and the bandwidth reserved for the tenant at links along these routes. In the following sections, we describe how the NM uses the above information to allocate tenant requests.

4.1 Cluster Allocation

A *virtual cluster* request r :<N, B> requires a virtual topology comprising N machines connected by links of bandwidth B to a virtual switch. In designing the allocation algorithm for such requests, we focus on tree-like physical network topologies; for instance, the multi-rooted trees used in today's datacenters and richer topologies like VL2 [1] and FatTree [28]. Such topologies are hierarchical and are recursively made of *sub-trees* at each level. For instance, with a three-level topology, the network is a collection of pods, pods comprise racks, and racks comprise hosts.

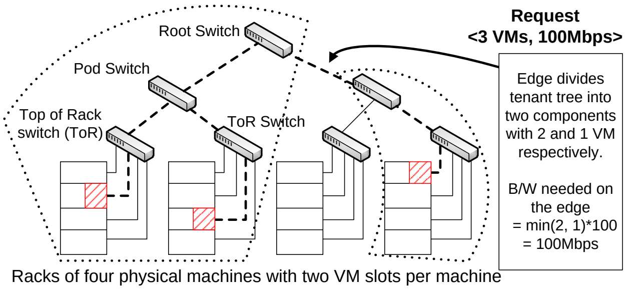

Request
<3 VMs, 100Mbps>
Root Switch
Pod Switch
Top of Rack switch (ToR)
ToR Switch
Edge divides tenant tree into two components with 2 and 1 VM respectively.
B/W needed on the edge = min(2, 1)*100 = 100Mbps
Racks of four physical machines with two VM slots per machine

Figure 5: An allocation for a cluster request r: <3, 100 Mbps>. Three VMs are allocated for the tenant at the highlighted slots. The dashed edges show the tenant tree T.

At a high level, the allocation problem involves allocating N empty VM slots to the tenant such that there is enough bandwidth to satisfy the corresponding virtual topology. We begin by characterizing the bandwidth requirements of an already allocated tenant on the underlying physical links. Further, we start with the assumption that the physical links connecting the tenant's N VMs form a simple tree T. This is shown in Figure 5. Section 4.4 relaxes the assumption. Note that the set of switches and links in T form a “distributed virtual switch” for the tenant. Given that the tenant's virtual switch has a bandwidth of $N * B$, a trivial yet inefficient solution is to reserve this bandwidth on each link in the tenant tree.

However, the actual bandwidth needed to be reserved is lower. Let's consider a link in T. As shown in Figure 5, removing this link from the tree leads to two components; if the first one contains m VMs, the other contains (N-m) VMs. The virtual topology dictates that a single VM cannot send or receive at rate more than B. Hence, traffic between these two components is limited to $\min(m, N-m) * B$. This is the *bandwidth required* for the tenant on this link.

For a *valid allocation*, the tenant's bandwidth requirement should be met on all links in the tenant tree. Hence, the *Virtual Cluster Allocation Problem* boils down to determining such valid allocations. An optimization version of this problem involves determining valid allocations that maximize Oktopus' future ability to accommodate tenant requests.

Allocation algorithm. Allocating *virtual cluster* re-

after “Paul the Oktopus” (German spelling), famous for being able to *predict* the outcome of football World Cup games.

quests on graphs with bandwidth-constrained edges is NP-hard [29]. We design a greedy allocation algorithm. The intuition here is that the number of tenant VMs that can be allocated to a sub-tree (a machine, a rack, a pod) is constrained by two factors. The first is the number of empty VM slots in the sub-tree. The second is the residual bandwidth on the physical link connecting the sub-tree to the rest of the network. This link should be able to accommodate the bandwidth requirements of the VMs placed inside the sub-tree. Given the number of VMs that can be placed in any sub-tree, the algorithm finds the smallest sub-tree that can fit all tenant VMs.

Below we introduce a few terms and explain the algorithm in detail. Each physical machine in the datacenter has K slots where VMs can be placed, while each link has capacity C. Further, $k_v \in [0, K]$ is the number of empty slots on machine v, while R_l is the residual bandwidth for link l. We begin by deriving constraints on the number of VMs that can be allocated at each level of the datacenter hierarchy. Starting with a machine as the base case, the number of VMs for request r that can be allocated to a machine v with outbound link l is given by the set M_v:

$$\begin{aligned} M_v \;=\; & \{m \in [0,\ \min(k_v, N)] \\ & \text{s.t.} \min(m, N-m) * B \leq R_l\} \end{aligned}$$

To explain this constraint, we consider a scenario where m ($< N$) VMs are placed at the machine v. As described earlier, the bandwidth required on outbound link l, $B_{r,l}$ is $\min(m, N-m)$*B. For a valid allocation, this bandwidth should be less than the residual bandwidth of the link. Note that in a scenario where all requested VMs can fit in v (i.e., $m = N$), all communication between the VMs is internal to the machine. Hence, the bandwidth needed for the request on the link is zero[4].

The same constraint is extended to determine the number of VMs that can be placed in sub-trees at each level, i.e., at racks at level 1, pods at level 2 and onwards. These constraints guide the allocation shown in Figure 6. Given the number of VMs that can be placed at each level of the datacenter hierarchy, the algorithm greedily tries to allocate the tenant VMs to the lowest level possible. To achieve this, we traverse the topology tree starting at the leaves (physical machines at level 0) and determine if all N VMs can fit (lines 2-10). Once the algorithm determines a sub-tree that can accommodate the VMs (line 5), it invokes the "Alloc" function to allocate empty slots on physical machines in the sub-tree to the tenant. While not shown in the algorithm, once the assignment is done, the bandwidth needed for the request is effectively "reserved" by updating the residual bandwidth for each link l as $R_l = R_l - B_{r,l}$.

```
Require: Topology tree T
Ensure: Allocation for request $r :< N, B >$
 1: $l = 0$ //start at level 0, i.e., with machines
 2: while true do
 3:   for each sub-tree $v$ at level $l$ of $T$ do
 4:     Calculate $M_v$        //$v$ can hold $M_v$ VMs
 5:     if $N \leq \max(M_v)$ then
 6:       Alloc($r$, $v$, $N$)
 7:       return true
 8:   $l = l + 1$ // move to higher level in $T$
 9:   if $l ==$ height($T$) then
10:     return false        //reject request

   //Allocate $m$ VM slots in sub-tree $v$ to request $r$
11: function Alloc($r$, $v$, $m$)
12: if (level($v$) ==0) then
13:   // Base case - $v$ is a physical machine
14:   Mark $m$ VM slots as occupied
15:   return $m$
16: else
17:   $count = 0$ //number of VMs assigned
18:   //Iterate over sub-trees of $v$
19:   for each sub-tree $w$ in $v$ do
20:     if $count < m$ then
21:       $count$ += Alloc($r$, $w$, $\min(m - count, \max(M_w))$)
22:   return $count$
```

Figure 6: Virtual Cluster Allocation algorithm

The fact that datacenter network topologies are typically oversubscribed (less bandwidth at root than edges) guides the algorithm's optimization heuristic. To maximize the possibility of accepting future tenant requests, the algorithm allocates a request while minimizing the bandwidth reserved at higher levels of the topology. This is achieved by packing the tenant VMs in the smallest sub-tree. Further, when multiple sub-trees are available at the same level of hierarchy, our implementation chooses the sub-tree with the least amount of residual bandwidth on the edge connecting the sub-tree to the rest of the topology. This preserves empty VM slots in other sub-trees that have greater outbound bandwidth available and hence, are better positioned to accommodate future tenants.

4.2 Oversubscribed Cluster Allocation

An oversubscribed cluster request, $r :<N, S, B, O>$, requires N VMs arranged in groups of size S. VMs within the same group are connected by links of bandwidth B to a virtual switch. Inter-group bandwidth is given by $B' = \frac{S*B}{O}$ (see Section 3.2).

Consider a request with three groups. As with the *virtual cluster*, any physical link in the tenant tree divides the tree into two components. Let g_i denote the VMs of group i that are in the first component, implying that the rest are in the second component ($S - g_i$). We observe that the bandwidth required by the request on the link is the sum of the bandwidth required by individual

[4] We assume that if the provider offers N slots per physical machine, the hypervisor can support N*B of internal bandwidth (within the physical machine).

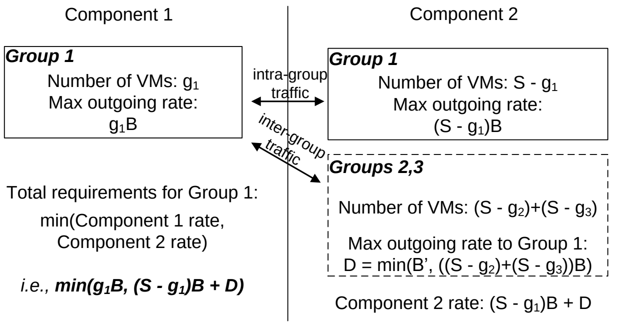


Figure 7: An oversubscribed cluster request with three groups. Figures illustrates bandwidth required by Group 1 VMs on a link dividing the tenant tree into two components.

groups. Focusing on the Group 1 VMs in the first component, their traffic on the link in question comprises the intra-group traffic to Group 1 VMs in the second component and inter-group traffic to VMs of Groups 2 and 3 in the second component. This is shown in Figure 7.

In the first component, Group 1 VMs cannot send (or receive) traffic at a rate more than $g_i * B$. In the second component, Group 1 VMs cannot receive (or send) at a rate more than $(S - g_i) * B$ while the rate for VMs of other groups cannot exceed the inter-group bandwidth B'. The rate of these other VMs is further limited by the aggregate bandwidth of the Group 2 and 3 members in the second component, i.e., $((S - g_2) + (S - g_3)) * B)$. Hence, as shown in the figure, the total bandwidth needed by Group 1 of request r on link l, $B_{r,1,l} = \min(g_1 * B, (S - g_1) * B + D)$. Finally, the total bandwidth required on the link is the sum across all three groups, i.e., $\sum_{i=1,3} B_{r,i,l}$.

Generalizing the analysis above, the bandwidth required for Group i on link l is given by

$$\begin{aligned} B_{r,i,l} &= \min(g_i * B, (S - g_i) * B + \\ &\quad + \min(B', \textstyle\sum_{j \neq i} (S - g_j) * B)). \end{aligned}$$

The bandwidth to be reserved on link l for request r is the sum across all the groups, i.e., $B_{r,l} = \sum_{i=1}^{P} B_{r,i,l}$. For the allocation to be valid, link l must have enough residual bandwidth to satisfy $B_{r,l}$. Hence, $B_{r,l} \leq R_l$ is the validity condition.

Allocation algorithm. The key insight guiding the algorithm is that allocating an oversubscribed cluster involves allocating a sequence of virtual clusters ($<S, B>$) for individual groups. This allows us to reuse the cluster allocation algorithm. Hence, the allocation for a request r proceeds one group at a time. Let's assume that groups 1 to (i-1) have already been allocated and we need to allocate VMs of group i. As with the cluster allocation algorithm, we derive constraints on the number of VMs for this group that can be assigned to each sub-tree. Consider a sub-tree with outbound link l already containing g_j members of group j, $j \in [1, i-1]$. Using the analysis above, the conditional bandwidth needed for the j^{th} group of request r on link l is:

$$CB_{r,j,l}(i-1) = \min(g_j * B, (S - g_j) * B + \min(B', E))$$

where,

$$E = \textstyle\sum_{k=1, k \neq j}^{i-1} (S - g_k) * B + \sum_{k=i}^{P} S * B.$$

This bandwidth is conditional since groups $i, \ldots, P$ remain to be allocated. We conservatively assume that all subsequent groups will be allocated outside the sub-tree and link l will have to accommodate the resulting inter-group traffic. Hence, if g_i members of group i were to be allocated inside the sub-tree, the bandwidth required by groups [1,i] on l is at most $\sum_{j=1}^{i} CB_{r,j,l}(i)$. Consequently, the number of VMs for group i that can be allocated to sub-tree v, designated by the set $M_{v,i}$, is:

$$\begin{aligned} M_{v,i} &= \{g_i \in [0,\ \min(k_v, S)] \\ &\quad \text{s.t. } \textstyle\sum_{j=1}^{i} CB_{r,j,l}(i) \leq R_l\}. \end{aligned}$$

Given the number of VMs that can be placed in sub-trees at each level of the datacenter hierarchy, the allocation algorithm proceeds to allocate VMs for individual groups using the algorithm in Figure 6. A request is accepted if all groups are successfully allocated.

4.3 Enforcing virtual networks

The NM ensures that the physical links connecting a tenant's VMs have sufficient bandwidth. Beyond this, Oktopus also includes mechanisms to *enforce* tenant virtual networks.

Rate limiting VMs. Individual VMs should not be able to exceed the bandwidth specified in the virtual topology. While this could be achieved using explicit bandwidth reservations at switches, the limited number of reservation classes on commodity switches implies that such a solution certainly does not scale with the number of tenants [27].

Instead, Oktopus relies on endhost based rate enforcement. For each VM on a physical machine, an *enforcement* module resides in the OS hypervisor. The key insight here is that given a tenant's virtual topology and the tenant traffic rate, it is feasible to calculate the rate at which pairs of VMs should be communicating. To achieve this, the enforcement module for a VM measures the traffic rate to other VMs. These traffic measurements from all VMs for a tenant are periodically sent to a tenant VM designated as the *controller VM*. The enforcement module at the controller then calculates the max-min fair share for traffic between the VMs. These rates are communicated back to other tenant VMs where the enforcement module uses per-destination-VM rate limiters to enforce them.

This simple design where rate computation for each tenant is done at a controller VM reduces control traffic. Alternatively, the enforcement modules for a tenant could use a gossip protocol to exchange their traffic rates, so that rate limits can be computed locally. We note that the enforcement modules are effectively achieving distributed rate limits; for instance, with a cluster request $<N, B>$, the aggregate rate at which the tenant's VMs can source traffic to a destination VM cannot exceed B. This is similar to the Distributed Rate Limiting (DRL) problem [30]. The authors discuss the trade-offs between accuracy and responsiveness versus the communication overhead in DRL; the same trade-offs apply here. Like Hedera [31], we perform centralized rate computation. However, our knowledge of the virtual topology makes it easier to determine the traffic bottlenecks. Further, our computation is tenant-specific which reduces the scale of the problem and allows us to compute rates for each virtual network independently. Section 5.4 shows that our implementation scales well imposing low communication overhead.

Tenants without virtual networks. The network traffic for tenants without guaranteed resources should get a (fair) share of the residual link bandwidth in the physical network. This is achieved using *two-level priorities*, and since commodity switches today offer priority forwarding, we rely on switch support for this. Traffic from tenants with a virtual network is marked as and treated as high priority, while other traffic is low priority. This, when combined with the mechanisms above, ensures that tenants with virtual networks get the virtual topology and the bandwidth they ask for, while other tenants get their fair share of the residual network capacity. The provider can ensure that the performance for fair share tenants is not too bad by limiting the fraction of network capacity used for virtual networks.

4.4 Design discussion

NM and Routing. Oktopus' allocation algorithms assume that the traffic between a given tenant's VMs is routed along a tree. However, they *do not rely on any assumptions about the underlying physical topology*, which may offer multiple paths between physical machines. The NM can ensure a routing tree for individual tenants using the following two approaches.

The spanning tree protocol ensures that traffic between machines in a layer2 domain is forwarded along a spanning tree. To scale up, datacenter networks typically comprise of multiple layer2 domains stitched up using a couple of layers of IP routers [32]. The IP routers are connected with a mesh of links that are used in active/passive mode or load balanced using ECMP. Given the amount of multiplexing over the mesh of links, they can be treated as a single aggregate link for bandwidth reservations. Hence, *with today's setup, the physical routing paths themselves form a tree and our assumption holds.* The NM only needs to infer this tree to determine the routing tree for any given tenant and hence, reserve bandwidth appropriately. This can be achieved using SNMP queries of the 802.1D-Bridge MIB on switches (products like Netview and OpenView support this) or through active probing [33].

Alternatively, the NM can control datacenter routing to actively build routes between tenant VMs, and recent proposals present backwards-compatible techniques to do just this. SecondNet [27] moves routing decisions from switches to a central controller that directly sends routing paths to endhosts. Endhosts use MPLS-based source routing to send traffic along these paths. Similarly, SPAIN [32] builds multiple VLANs over the underlying physical topology and allows endhosts to specify the VLAN to use for their packets. The Oktopus NM can adopt either approach to build tenant-specific routing trees on top of rich physical topologies, and direct the OS hypervisor to do the source route marking (for SecondNet) or VLAN tagging (for SPAIN).

Failures. Failures of physical links and switches in the datacenter will impact the virtual topology for tenants whose routing tree includes the failed element. With today's setup, providers are not held responsible for physical failures and tenants end up paying for them [16]. Irrespective, our allocation algorithms can be extended to determine the tenant VMs that need to be migrated, and reallocate them so as to satisfy the tenant's virtual topology. For instance, with the cluster request, the failed edge divides the tenant's routing tree into two components. If the NM cannot find alternate links with sufficient capacity to connect the two components, it will reallocate the VMs present in the smaller component.

Further, such an extended allocation scheme can also accommodate tenant contraction and expansion wherein tenants want to decrease or increase the size of their virtual topology in an incremental fashion.

5. EVALUATION

We evaluate two aspects of Oktopus. First, we use large-scale simulations to quantify the benefits of providing tenants with bounded network bandwidth. Second, we show that the Oktopus NM can deal with the scale and churn posed by datacenters, and benchmark our implementation on a small testbed.

5.1 Simulation setup

Since our testbed is restricted to 25 machines, we developed a simulator that coarsely models a multi-tenant datacenter. The simulator uses a three level, tree-like network topology (as described in Section 4). The results in the following sections involve a datacenter with 16,000 physical machines and 4 VMs per machine, resulting in a total of 64,000 VMs.

Given the poor performance of packet level simulators at such scales [31], our simulator models flow level communication between tenant VMs. For today's setup, the per-flow bandwidth is calculated according to max-min fairness, and a flow's rate is its fair share across the most bottlenecked physical link it traverses. For tenants with virtual networks, flow rates are determined by a similar fair share calculation over *the tenant's virtual topology.*

Allocation. We implemented the allocation algorithms presented in Section 4. For today's setup, we also implemented a locality-aware allocation algorithm. The algorithm greedily allocates tenant VMs as close to each other as possible.

Tenant workload. We adopt a broad yet realistic model for jobs/applications run by tenants. Tenant jobs comprise computation and network communication. To this effect, each tenant job is modeled as a combination of the minimum compute time for the job (T_c) and a set of network flows between tenant VMs. We start with a simple communication pattern wherein each tenant VM is a source and a destination for one flow, and all flows are of uniform length (L). A job is complete when both the computation and the network flows finish. Hence, the completion time for a job, $T = \max(T_c, T_n)$, where T_n is the time for the last flow to finish. This reflects real-world workloads. For instance, with MapReduce, the job completion time is heavily influenced by the last shuffle flow and the slowest task [15].

This naive workload model was deliberately chosen; the job compute time T_c abstracts away the non-network resources required and allows us to determine the tenant's "network requirements". Since tenants pay based on the time they occupy VMs and hence, their job completion time, tenants can minimize their cost by ensuring that their network flows do not lag behind the computation, i.e., $T_n \leq T_c$. With the model above, the network bandwidth needed by tenant VMs to achieve this is $B = \frac{L}{T_c}$.

Tenant requests. To allow direct comparisons, we ensure that any given tenant request with the above workload can be expressed as a status quo request and as a virtual network request. Tenant requests have the form of $<N>$, $<N, B>$ and $<N, S, B, O>$ to represent today's setup, the cluster and oversubscribed cluster respectively. For the latter case, to ensure that the underlying job workload matches the request specification, the number of inter-group flows is proportional to the oversubscription factor O. For example, if O=10, on average $\frac{N}{10}$ inter-group flows are generated per request.

Simulation breadth. Since there are currently no available datasets describing job bandwidth requirements to guide our workload, our evaluation explores the entire space for most parameters of interest in today's datacenters; these include tenant bandwidth requirements, datacenter load, and physical topology oversubscription. This is not only useful for completeness, but, further, provides evidence of Oktopus's performance at the extreme points. Finally, for parameters that can be inferred through our datasets in Section 2, such as job sizes (i.e., N), we use the corresponding values or distributions.

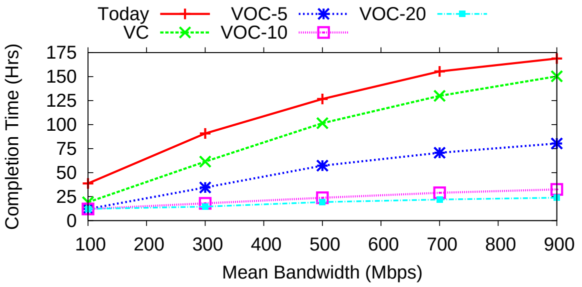


Figure 8: Completion time for a batch of 10,000 tenant jobs today and with various virtual network abstractions.

5.2 Production datacenter experiments

We first consider a scenario involving a large batch of tenant jobs to be allocated and run in the datacenter. The experiment is representative of the workload observed in production datacenters running data-analytics jobs from multiple groups/services. We compare the throughput achieved with virtual network abstractions against the status quo.

In our experiments, the number of VMs (N) requested by each tenant is exponentially distributed around a mean of 49. This is consistent with what is observed in production and cloud datacenters [14]. For oversubscribed cluster requests, the tenant VMs are arranged in $\sqrt{N}$ groups each containing $\sqrt{N}$ VMs, ensuring that the number of groups do not grow linearly with N. We begin with a physical network with 10:1 oversubscription, a conservative value given the high oversubscription of current data center networks [1], and 4 VMs per physical machine. We simulate the execution of a batch of 10,000 tenant jobs with varying mean bandwidth requirements for the jobs. To capture the variability in network intensiveness of jobs, their bandwidth requirements are taken from an exponential distribution around the mean. The job scheduling policy is the same throughout– jobs are placed in a FIFO queue, and once a job finishes, the topmost job(s) that can be allocated are allowed to run.

Job completion time. Figure 8 plots the time to complete all jobs with different abstractions for tenants– *today*'s setup, virtual cluster (VC), and virtual oversubscribed cluster with varying oversubscription ratio (VOC-10 refers to oversubscribed clusters with O=10). The figure shows that for any given approach, the com-

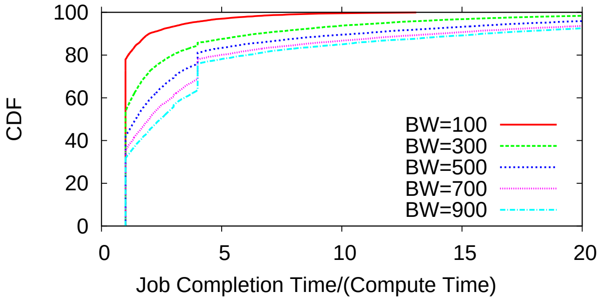


Figure 9: With today's setup, job duration is extended.

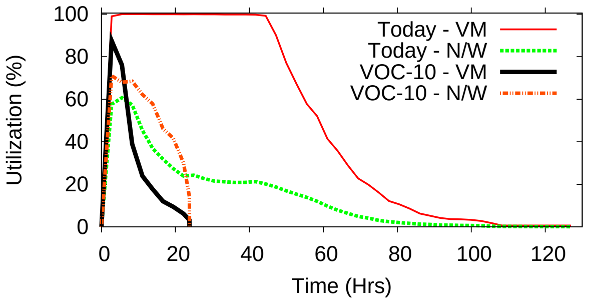


Figure 10: VM and network utilization today and with VOC-10.

pletion time increases as the mean bandwidth requirement increases (i.e., jobs become network intensive).

In all cases, *virtual clusters provide significant improvement over the completion time observed today.* For oversubscribed clusters, the completion time depends on the oversubscription ratio. The completion time for VOC-2, omitted for clarity, is similar to that of *virtual cluster*. With VOC-10, the completion time at 500 Mbps is 18% (*6 times less*) of today's setup (31% with 100 Mbps). Note that increasing O implies greater locality in the tenant's communication patterns. This allows for more concurrent tenants and reduces completion time which, in turn, improves datacenter throughput. However, the growth in benefits with increasing oversubscription diminishes, especially beyond a factor of 10.

Beyond improving datacenter throughput, providing tenants with virtual networks has other benefits. It ensures that network flows comprising a job do not lag behind computation. Hence, a tenant job, once allocated, takes the minimum compute time T_c to complete. However, with today's setup, varying network performance can cause the completion time for a job to exceed T_c. Figure 9 plots the CDF for the ratio of today's job completion time to the compute time and shows that tenant jobs can be stretched much longer than expected. With BW=500 Mbps, the completion time for jobs is 1.42 times the compute time at the median (2.8 times at the 75th percentile). Such performance unpredictability is highly undesirable and given the iterative program/debug/modify development, hurts programmer productivity [15].

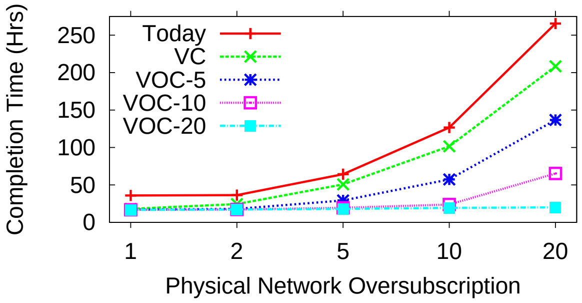


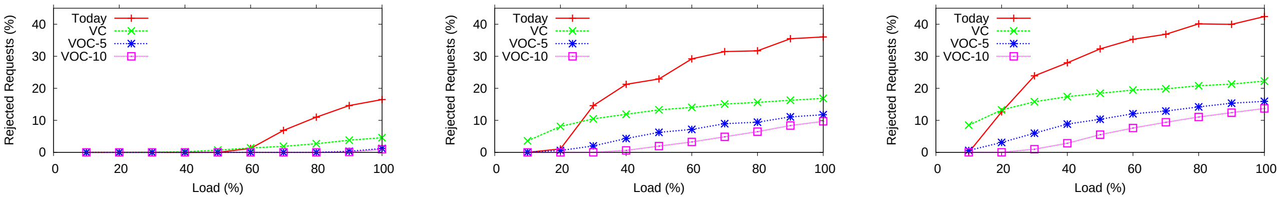
Figure 11: Completion time increases with physical oversub. Mean BW = 500 Mbps

Utilization. To understand the poor performance with today's setup, we look at the average VM and network utilization over the course of one experiment. This is shown in Figure 10. Today, the network utilization remains low for a majority of the time. This is because the allocation of VMs, though locality aware, does not account for network demands causing contention. Thus, tenant VMs wait for the network flows to finish and hurt datacenter throughput. As a contrast, with oversubscribed cluster (VOC-10), the allocation is aware of the job's bandwidth demands and hence, results in higher network utilization.

We repeated the experiments above with varying parameters. Figure 11 shows how the completion time varies with the physical network oversubscription. We find that *even when the underlying physical network is not oversubscribed as in* [1,28]*, virtual networks can reduce completion time (and increase throughput) by a factor of two*[5]. Further, increasing the virtual oversubscription provides greater benefits when the physical oversubscription is larger. Similarly, increasing the tenant size (N) improves the performance of our abstractions relative to today since tenant traffic is more likely to traverse core network links. We omit these results due to space constraints.

Mis-estimating network requirements. In the experiment above, the bandwidth requested is exactly what is required to finish the tenant's flows on time. In practice, a tenant's characterization of her network requirements is likely to be imprecise. Hence, we introduce "errors in job demands"; an error of 50% implies that the bandwidth requested by tenants is normally distributed around the actual bandwidth needed with a standard deviation of 50%. This means that for ~35% of tenants, the bandwidth requested is $<\frac{1}{2}$x or $>$2x the actual bandwidth. Note that when tenants ask for less

[5] Since there are 4 VMs per machine, flows for a VM can still be network bottlenecked at the outbound link.



(a) Mean BW 100 Mbps (b) Mean BW 500 Mbps (c) Mean BW 900 Mbps

Figure 13: Percentage of rejected tenant requests with varying datacenter load and varying mean tenant bandwidth requirements. At load>20%, virtual networks allow more requests to be accepted.

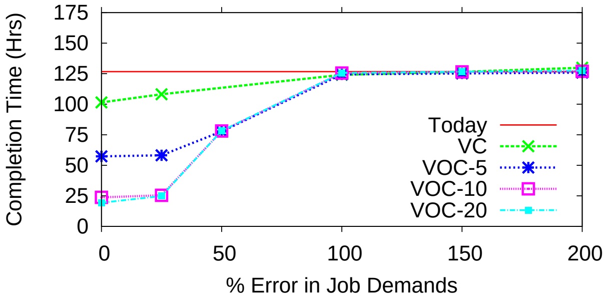


Figure 12: Error in job demands impacts completion time. Mean BW = 500 Mbps.

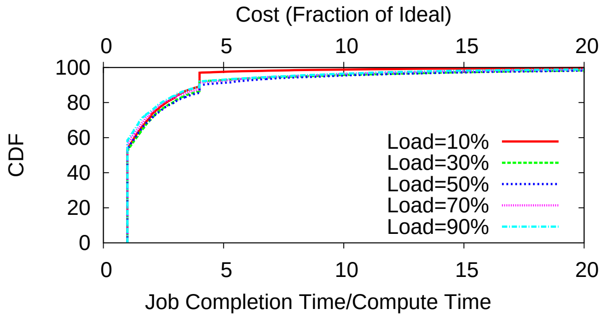


Figure 14: CDF for increase in completion time and cost (upper X-axis) with today's setup. Mean BW = 500Mbps

bandwidth than needed, their flows will lag behind their computation while when they ask for more, they will under-utilize the underlying network.

Figure 12 shows that even with "loose" specifications of tenant demands, *the completion time, while increasing for large errors, still does not exceed today's completion time* (which is independent of errors in job demands). Further, since the bandwidth that tenants demand is restricted to [50, 1000 Mbps], increasing the error beyond 100% does not impact virtual network completion time.

5.3 Cloud datacenter experiments

The experiments in the previous section involved a static set of jobs. We now introduce tenant dynamics with tenant requests arriving over time. This is representative of cloud datacenters. By varying the tenant arrival rate, we vary the load imposed in terms of the number of VMs. Assuming Poisson tenant arrivals with a mean arrival rate of λ, the load on a datacenter with M total VMs is $\frac{\lambda N T_c}{M}$, where N is the mean request size and T_c is the mean compute time. A request may be rejected if it cannot be allocated. We simulate the arrival and execution of 10,000 tenant requests with varying mean bandwidth requirements for the tenant jobs.

Rejected requests. Figure 13 shows that *only at very low loads, today's setup is comparable with the virtual abstractions in terms of rejected requests, despite the fact that virtual abstractions explicitly reserve the bandwidth requested by tenants.* At low loads, requests arrive far apart in time and, thus, they can be always allocated even though today's setup prolongs job completion. As the load increases, today's setup rejects far more requests. For instance, at 70% load (Amazon EC2's operational load [34]) and bandwidth of 500Mbps, 31% of requests are rejected today as compared to 15% of VC requests and only 5% of VOC-10 requests.

Tenant costs and provider revenue. Today's cloud providers charge tenants based on the time they occupy their VMs. Assuming a price of k dollars per-VM per unit time, a tenant using N VMs for time T pays kNT dollars. This implies that *while intra-cloud network communication is not explicitly charged for, it is not free* since poor network performance can prolong tenant jobs and hence, increase their costs. Figure 14 shows the increase in tenant job completion times and the corresponding increase in tenant costs (upper X-axis) today. For all load values, many jobs finish later and cost more than expected– the cost for 25% tenants is more than 2.3 times their ideal cost had the network performance been sufficient (more than 9.2 times for 5% of the tenants).

The fraction of tenant requests that are accepted and the costs for accepted requests govern the provider revenue. Figure 15 shows the provider revenue when tenants use virtual networks relative to today's revenue. At low load, the provider revenue is reduced since the use of virtual networks ensures that tenant jobs finish faster and they pay significantly less. However, as the load

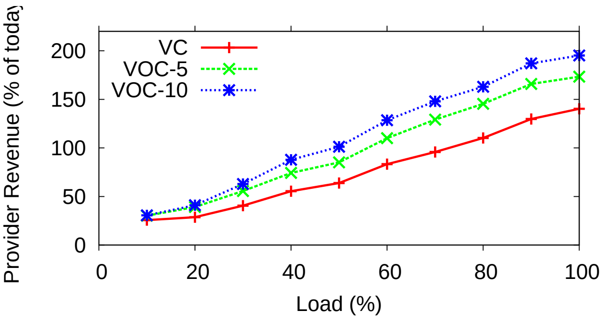


Figure 15: Provider revenue with virtual network abstractions. Mean BW = 500Mbps

increases, the provider revenue increases since virtual network approaches allow more requests to be accepted, even though individual tenants pay less than today. For efficiency, providers like Amazon operate their datacenters at an occupancy of 70-80% [34]. Hence, *for practical load values, virtual networks not only allow tenants to lower their costs, but also increase provider revenue!* Further, this estimation ignores the extra tenants that may be attracted by the guaranteed performance and reduced costs.

Charging for bandwidth. Providing tenants with virtual networks opens the door for explicitly charging for network bandwidth. This represents a more fair charging model since a tenant should pay more for a *virtual cluster* with 500Mbps than one with 100Mbps. Here, we explore the following simple charging model. Apart from paying for VM occupancy (k_v), tenants also pay a bandwidth charge of $k_b \frac{\$}{\text{bw*unit-time}}$. Hence, a tenant using a virtual cluster $<N, B>$ for time T pays $NT(k_v + k_bB)$.

Such a charging model presents an opportunity to redress the variability in provider revenue observed above. To this effect, we performed the following analysis. We used current Amazon EC2 prices to determine k_v and k_b for each virtual network abstraction so as to maintain provider revenue neutrality, i.e., the provider earns the same revenue as today.[6] We then determine the ratio of a tenant's cost with the new charging model to the status quo cost. The median tenant cost is shown in Figure 16. We find that except at low loads, *virtual networks can ensure that providers stay revenue neutral and tenants pay significantly less than today while still getting guaranteed performance.* For instance, with a mean bandwidth demand of 500 Mbps, Figure 16 shows that tenants with virtual clusters pay 68% of today at moderate load and 37% of today at high load (31% and 25% respectively with VOC-10).

The charging model above can be generalized from linear bandwidth costs to $NT(k_v+k_bf(B))$, where f is a bandwidth charging function. We repeated the analysis with other bandwidth functions ($B^{\frac{3}{2}}, B^2$) and obtained similar results.

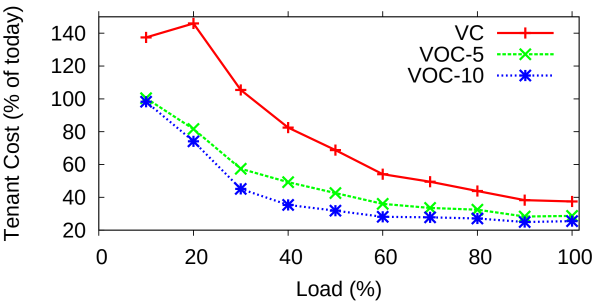


Figure 16: Relative tenant costs based on the bandwidth charging model while maintaining provider revenue neutrality (mean BW=500 Mbps). For load>20%, tenants pay substantially less than today.

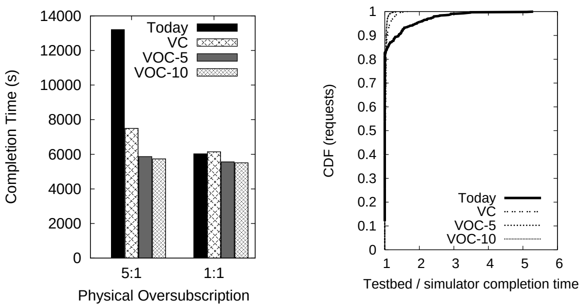
5.4 Implementation and Deployment

Our Oktopus implementation follows the description in Section 4. The NM maintains reservations across the network and allocates tenant requests in an on-line fashion. The enforcement module on individual physical machines implements the rate computation and rate limiting functionality (Section 4.3). For each tenant, one of the tenant's VMs (and the corresponding enforcement module) acts as a controller and calculates the rate limits. Enforcement modules then use the Windows Traffic Control API [35] to enforce local rate limits on individual machines.

Scalability. To evaluate the scalability of the NM, we measured the time to allocate tenant requests on a datacenter with 10^5 endhosts. Over 10^5 requests, the median allocation time is 0.35ms with a 99^{th} percentile of 508ms. Note that this only needs to be run when a tenant is admitted, and hence, the NM can scale to large datacenters.

The rate computation overhead depends on the tenant's communication pattern. Even for a tenant with 1000 VMs (two orders of magnitude more than mean tenant size today [14]) and a *worst-case* scenario where all VMs communicate with all other VMs, the computation takes 395ms at the 99^{th} percentile. With a typical communication pattern [36], 99^{th} percentile computation time is 84ms. To balance the trade-off between accuracy and responsiveness of enforcement and the communication overhead, our implementation recomputes rates every 2 seconds. For a tenant with 1000 VMs and *worst-case* all-to-all communication between the VMs, the controller traffic is 12 Mbps (∼1 Mbps with a typical communication pattern). Hence, the enforcement

[6] For Amazon EC2, small VMs cost 0.085$/hr. Sample estimated prices in our experiments are at 0.04$/hr for k_v, and 0.00016$ /GB for k_b.



(a) Completion Time (b) Cross Validation

Figure 17: Testbed experiments show that virtual networks provide performance gains and validate our simulator.

module imposes low overhead.

Deployment. We deployed Oktopus on a testbed with 25 endhosts arranged in five racks. Each rack has a Top-of-Rack (ToR) switch which is connected to a root switch. Each interface is 1 Gbps. Hence, the testbed has a two-tier tree topology with a physical oversubscription of 5:1. All endhosts are Dell Precision T3500 servers with a quad core Intel Xeon 2.27GHz processor and 4GB RAM, running Windows Server 2008 R2. Given our focus on quantifying the benefits of Oktopus abstractions, instead of allocating VMs to tenants, we simply allow their jobs to run on the host OS. However, we retain the limit of 4 jobs per endhost, resulting in a total of 100 VM or job slots.

We repeat the experiments from Section 5.2 on the testbed and determine the completion time for a batch of 1000 tenant jobs (mean tenant size N is scaled down to 9). As before, each tenant job has a compute time (but no actual computation) and a set of TCP flows associated with it. Figure 17(a) shows that virtual clusters reduce completion time by 44% as compared to today (57% for VOC-10). We repeated the experiment with all endhosts connected to one switch (hence, no physical oversubscription). The bars on the right in Figure 17(a) show that virtual clusters match today's completion time while VOC-10 offers a 9% reduction. Since the scale of these experiments is smaller (smaller topology and tenants), virtual networks do not have much opportunity to improve performance and the reduction in completion time is less significant. However, tenant jobs still get guaranteed network performance and hence, predictable completion times.

Cross-validation. We replayed the same job stream in our simulator and for each tenant request, we determined the ratio of the completion time on the testbed and the simulator. Figure 17(b) shows that for the vast majority of jobs, the completion time in the simulator matches that on the testbed. Some divergence results from the fact that network flows naturally last longer in the live testbed than in the simulator which optimally estimates the time flows take. We note that jobs that last longer in the testbed than the simulator occur more often with today's setup than with virtual networks. This is because today's setup results in more network contention which, in turn, causes TCP to not fully utilize its fair share. Overall, the fact that the same workload yields similar performance in the testbed as in the simulator validates our simulation setup and strengthens our confidence in the results presented.

6. RELATED WORK

The increasing prominence of multi-tenant datacenters has prompted interest towards datacenter network virtualization. Seawall [14] and NetShare [37] share the network bandwidth among tenants based on weights. The resulting proportional bandwidth distribution leads to efficient multiplexing of the underlying infrastructure; yet, in contrast to Oktopus, tenant performance still depends on other tenants. SecondNet [27] provides pairwise bandwidth guarantees where tenant requests can be characterized as $<N, [B_{ij}]_{N\times N}>$; $[B_{ij}]_{N\times N}$ reflects the complete pairwise bandwidth demand matrix between VMs. With Oktopus, we propose and evaluate more flexible virtual topologies that balance the tradeoff between tenant demands and provider flexibility.

Duffield et al. [29] introduced the hose model for wide-area VPNs. The hose model is akin to the *virtual cluster* abstraction; however, the corresponding allocation problem is different since the physical machines are fixed in the VPN setting while we need to choose the machines. Our argument regarding the greater flexibility of the *virtual cluster* over a clique virtual topology is, in part, guided by the flexibility of the hose model over the pipe model in VPN settings. Other allocation techniques like simulated annealing and mixed integer programming have been explored as part of testbed mapping [38] and virtual network embedding [39]. These efforts focus on allocation of arbitrary (or, more general) virtual topologies on physical networks which hampers their scalability and restricts them to small physical networks (O(10^2) machines).

The use of network virtualization as an enabler for next-generation network protocols has been studied, both in the wide-area [40] and local production networks [41]. Beyond this, there is a large body of work focusing on bandwidth guarantees (e.g., IntServ) and differentiated services (e.g., DiffServ) in the Internet. Sun's Crossbow network stack [42] provides virtual NICs with configurable local bandwidth limits. that prevent interference from co-located VMs. IBM iCorpMaker and HP Utility Data Center (UDC) provide support for virtual circuits and bandwidth reservation for several classes of traffic. If available, Oktopus can use these technologies to enforce virtual networks.

7. CONCLUDING REMARKS

This paper presents virtual network abstractions that allow tenants to expose their network requirements. This enables a *symbiotic* relationship between tenants and providers; tenants get a predictable environment in shared settings while the provider can efficiently match tenant demands to the underlying infrastructure, without muddling their interface. Our experience with Oktopus shows that the abstractions are practical, can be efficiently implemented and provide significant benefits.

Our abstractions, while emulating the physical networks used in today's enterprises, focus on a specific metric– inter-VM network bandwidth. Tenants may be interested in other performance metrics, or even non-performance metrics like reliability. Examples include bandwidth to the storage service, latency between VMs and failure resiliency of the paths between VMs. In this context, virtual network abstractions can provide a succinct means of information exchange between tenants and providers.

Another interesting aspect of virtual networks is cloud pricing. Our experiments show how tenants can implicitly be charged for their internal traffic. By offering bounded network resources to tenants, we allow for *explicit and more fair* bandwidth charging. More generally, charging tenants based on the characteristics of their virtual networks eliminates hidden costs and removes a key hindrance to cloud adoption. This, in effect, could pave the way for multi-tenant datacenters where tenants can pick the trade-off between the performance of their applications and their cost.

8. REFERENCES

[1] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *Proc. of ACM SIGCOMM*, 2009.

[2] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: comparing public cloud providers," in *Proc. of conference on Internet measurement (IMC)*, 2010.

[3] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," in *Proc. of VLDB*, 2010.

[4] D. Mangot, "Measuring EC2 system performance," May 2009, http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_revealed.

[5] A. Giurgiu, "Network performance in virtual infrastrucures," Feb. 2010, http://staff.science.uva.nl/~delaat/sne-2009-2010/p29/presentation.pdf.

[6] Michael Armburst et al., "Above the Clouds: A Berkeley View of Cloud Computing," University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009.

[7] B. Craybrook, "Comparing cloud risks and virtualization risks for data center apps," Jan. 2011, http://searchdatacenter.techtarget.com/tip/Comparing-cloud-risks-and-virtualization-risks-for-data-center-apps.

[8] A. Iosup, N. Yigitbasi, and D. Epema, "On the Performance Variability of Production Cloud Services," Delft University of Technology, Tech. Rep. PDS-2010-002, Jan. 2010.

[9] D. Kossmann, T. Kraska, and S. Loesing, "An Evaluation of Alternative Architectures for Transaction Processing in the Cloud," in *Proc. of international conference on Management of data (SIGMOD)*, 2010.

[10] M. Zaharia, A. Konwinski, A. D. Joseph, Y. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *Proc. of USENIX OSDI*, 2008.

[11] E. Walker, "Benchmarking Amazon EC2 for high-performance scientific computing," *IEEE login*, vol. 33, 2008.

[12] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proc. of ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[13] "Amazon Cluster Compute," Jan. 2011, http://aws.amazon.com/ec2/hpc-applications/.

[14] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Sharing the Datacenter Network," in *Proc. of ACM/USENIX NSDI*, 2011.

[15] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the Outliers in Map-Reduce Clusters using Mantri," in *Proc. of USENIX OSDI*, 2010.

[16] H. Wang, Q. Jing, S. Jiao, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed Systems Meet Economics: Pricing in the Cloud," in *Proc. of USENIX HotCloud*, 2010.

[17] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *Proc. of IEEE Infocom*, 2010.

[18] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proc. of OSDI*, 2004.

[19] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *Proc. of EuroSys*, 2007.

[20] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: easy and efficient parallel processing of massive data sets," in *Proc. of VLDB*, 2008.

[21] S. Kandula, J. Padhye, and P. Bahl, "Flyways To De-Congest Data Center Networks," in *Proc. of HotNets*, 2005.

[22] c-Through: Part-time Optics in Data Centers, "Guohui Wang and David G. Andersen and Michael Kaminsky and Konstantina Papagiannaki and T. S. Eugene Ng and Michael Kozuch and Michael Ryan," in *Proc. of ACM SIGCOMM*, 2010.

[23] X. Meng, V. Pappas, and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," in *Proc. of IEEE Infocom*, 2010.

[24] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," in *Proceedings of ACM SIGCOMM*, 2010.

[25] "Amazon EC2 Spot Instances," http://aws.amazon.com/ec2/spot-instances/.

[26] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach.* Elseview Science, 2003.

[27] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. of ACM CoNext*, 2010.

[28] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of ACM SIGCOMM*, 2008.

[29] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive, "A flexible model for resource management in virtual private networks," in *Proc. of ACM SIGCOMM*, 1999.

[30] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," in *Proc. of ACM SIGCOMM*, 2007.

[31] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for

Data Center Networks," in *Proc. of USENIX NSDI*, 2010.

[32] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies." in *Proc of NSDI*, 2010.

[33] R. Black, A. Donnelly, and C. Fournet, "Ethernet Topology Discovery without Network Assistance," in *Proc. of ICNP*, 2004.

[34] "Amazon's EC2 Generating 220M," Jan. 2011, http://cloudscaling.com/blog/cloud-computing/amazons-ec2-generating-220m-annually.

[35] "Traffic Control API," Jan. 2011, http://msdn.microsoft.com/en-us/library/aa374468%28v=VS.85%29.aspx.

[36] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proc. of ACM IMC*, 2009.

[37] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," University of California, San Deigo, Tech. Rep. CS2010-0957, May 2010.

[38] R. Ricci, C. Alfeld, and J. Lepreau, "A Solver for the Network Testbed Mapping problem," *SIGCOMM Comput. Commun. Rev.*, vol. 33, 2003.

[39] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, 2008.

[40] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *Proc. of SIGCOMM*, 2006.

[41] R. Sherwood, "FlowVisor," Jan. 2011, http://www.openflowswitch.org/wk/index.php/FlowVisor.

[42] "Crossbow: Network Virtualization and Resource Control," Jan. 2011, http://hub.opensolaris.org/bin/view/Project+crossbow/WebHome.