THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

CloudMirror: Application-Aware Bandwidth Reservations in the Cloud

Citation for published version:
Lee, J, Lee, M, Popa, L, Turner, Y, Banerjee, S, Sharma, P & Stephenson, B 2013, CloudMirror: Application-Aware Bandwidth Reservations in the Cloud. in *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing.* Usenix, Berkeley, CA, pp. 1-6. <https://www.usenix.org/conference/hotcloud13/workshop-program/presentations/Lee>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 22. May. 2024

CloudMirror: Application-Aware Bandwidth Reservations in the Cloud

Jeongkeun Lee[+], Myungjin Lee[*], Lucian Popa[+], Yoshio Turner[+], Sujata Banerjee[+],
Puneet Sharma[+], Bryan Stephenson[!]
[+]HP Labs, [*]University of Edinburgh, [!]HP Enterprise Services

Abstract

Cloud computing providers today do not offer guarantees for the network bandwidth available in the cloud, preventing tenants from running their applications predictably. To provide guarantees, several recent research proposals offer tenants a virtual cluster abstraction, emulating physical networks. Whereas offering dedicated virtual network abstractions is a significant step in the right direction, in this paper we argue that the abstractions exposed to tenants should aim to model tenant application structures rather than aiming to mimic physical network topologies. The fundamental problem in providing users with dedicated network abstractions is that the communication patterns of real applications do not typically resemble the rigid physical network topologies. Thus, the virtual network abstractions often poorly represent the actual communication patterns, resulting in overprovisioned/wasted network resources and underutilized computational resources.

We propose a new abstraction for specifying bandwidth guarantees, which is easy to use because it closely follows application models; our abstraction specifies guarantees as a graph between application components. We then propose an algorithm to efficiently deploy this abstraction on physical clusters. Through simulations, we show that our approach is significantly more efficient than prior work for offering bandwidth guarantees.

1 Introduction

Today, cloud computing providers guarantee CPU, memory and storage resources to tenant virtual machines (VMs), but do not offer guarantees for the network bandwidth available in the cloud.[1] The lack of network bandwidth guarantees prevents tenants from achieving predictable performance for their applications [2]. Predictability, however, is a key requirement for the ability to migrate applications to the cloud. For example, many user-facing applications have strict response-time requirements [3], and may also require high network bandwidths, *e.g.,* [4].[2] When these applications share a cloud datacenter network with bandwidth-intensive batch applications (*e.g.,* MapReduce), this interaction can severely hurt the performance of the user-facing applications.

A key challenge in providing bandwidth guarantees is to offer an effective *abstraction* to tenants for expressing the bandwidth guarantees required for an application. The research community has recently recognized the need for bandwidth guarantees in the cloud, and several projects have proposed various abstractions and connectivity models for bandwidth guarantees [6, 2, 7, 8, 9, 10, 11]. Most of these proposals aim to model a dedicated virtual network topology, resembling a physical topology, for each tenant. A tree-shaped cluster abstraction, *e.g.,* either a single-level tree, known as the hose model, or a multi-level tree (see [2]), is most commonly used [6, 2, 8, 9, 10, 11]. This abstraction directly maps to today's tree-shaped physical network topologies, and is easy to understand.

In this paper, we argue that the abstractions exposed to tenants should *aim to model tenant application structures rather than aiming to model physical network topologies*. The fundamental problem in providing users with dedicated network abstractions is that the communication patterns of real applications do not typically resemble the rigid physical network topologies. Thus, the virtual network abstractions often poorly represent the actual communication patterns, resulting in overprovisioned/wasted network resources and underutilized computational resources (see §2 and §5). Offering more complex virtual network abstractions (such as multi-layered trees) in an attempt to reduce this inefficiency is only partially effective, and significantly complicates the job of tenants to map their applications onto the virtual network topologies.

Our solution to this problem is *CloudMirror*, a framework that comprises a novel abstraction for expressing guarantees and an efficient VM placement algorithm that takes advantage of this abstraction. CloudMirror's abstraction, called *Tenant Application Graph (TAG)*, aims to mirror the structure of the application being deployed by the tenant. Specifically, with a TAG abstraction, a tenant specifies a *graph of bandwidth*

[1] Amazon EC2 has very weak network SLAs; its Cluster Networking promises 'high-bandwidth' but there is no guarantee and no way to specify a custom bandwidth demand [1].

[2] Twitter's real-time big-data computation system, Storm, can process over 10^6 tuples/second per VM [4]; this requires 240 Mbps network bandwidth per VM, assuming 30 byte tuples [5]).

guarantees between the application tiers. Since TAG models directly capture the application structure, they are both easy to use and accurately represent the communication patterns of applications. Furthermore, TAG models naturally accommodate load balancing between application tiers, dynamic "flexing" of the application size based on load [12], and middleboxes (see §3).

CloudMirror's VM placement algorithm leverages the TAG model to efficiently place VMs on tree-shaped datacenter topologies (§4). Our evaluation (§5) shows that compared to our model and placement algorithm, Oktopus [2] requires 60% more bandwidth for the same application set. In turn, this results in hosting more tenants on the same network or deploying a smaller network for the same workload.

2 Shortcomings of Prior Models

Most prior research efforts on cloud networking have focused on batch processing applications like MapReduce and Pregel. These data and network-intensive applications typically employ a simple communication pattern, *e.g.,* all mappers send data to all reducers. Still, the cloud hosts a wide range of applications beyond batch jobs, with different communication patterns. For instance, cloud datacenters host many user-facing applications and sophisticated enterprise applications composed of many tiers with complex traffic interactions [13]. Consider the following two illustrative example applications.

Fig. 1(a) shows a simple example of a three-tiered application with a frontend web tier, a business logic tier, and a backend database tier. Each tier contains multiple VMs and the edges of the communication graph are annotated with the bandwidth requirements between tiers.

The second example is a real-time data analytic application shown in Fig. 2(a), implemented using Storm [4]. Storm is a platform used by many companies for online machine learning, continuous computation on data streams, *etc*. Storm applications have two types of components: "spouts", which are similar to mappers in MapReduce, and "bolts", which represent both a mapper and a reducer. [3]

As cloud datacenters aggressively consolidate diverse workloads onto a shared fabric to maximize resource efficiency, it is critical to adequately represent all the different types of communication patterns (such as the above examples), and not only the batch processing jobs.

Next, we discuss the three most commonly used abstractions: hose model, VOC model and pipe model.

- **Hose Model:** In the hose model [6, 2, 8, 10, 11], all VMs are connected to a central (virtual) switch by a dedicated link (hose) having a minimum bandwidth guarantee. To better match application requirements, we consider a generalized hose model [6] where each VM can have a different bandwidth guarantee (unlike Oktopus [2]).

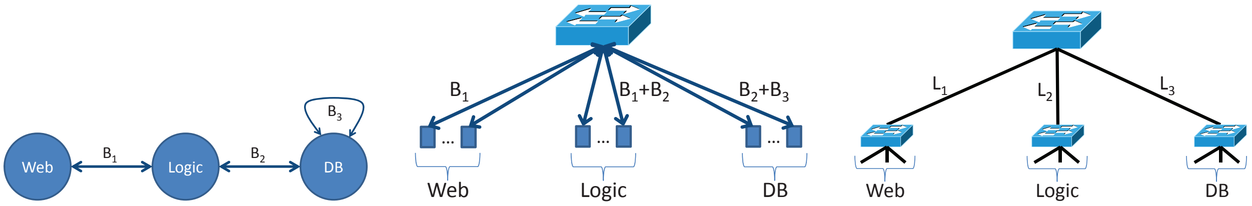
While this model is simple to derive, it can be severely inefficient. Consider the example in Fig. 1(a) and assume that B_1 represents the typical bandwidth demand between one VM from the web tier and one VM for the business logic tier. B_2 is defined similarly, while B_3 is the bandwidth demand between two database VMs (*e.g.,* used to ensure the consistency of the database). For simplicity we assume an equal number of VMs in each tier and equal bandwidth requirements in both directions. Also, we ignore the Internet bandwidth requirement to the web tier.

Fig. 1(b) presents the hose model guarantees for the example in Fig. 1(a). The fundamental problem is that the hose model does not accurately capture the traffic pattern between application tiers. More concretely, suppose that each server tier is deployed on a separate sub-tree of the physical network as shown in Fig. 1(c). To satisfy the hose model, the bandwidth that must be reserved on link L_3 for each database VM would be B_2+B_3. (We assume here that $B_2+B_3 < B_1+B_1+B_2$, and so the minimum that needs to be reserved on link L_3 is B_2+B_3 rather than $2B_1$+B_2.) However, the hose model hides the fact that B_3 is used solely for the communication within the DB tier rather than for communication with other tiers. Thus, the application does not actually need the full guarantees (B_2+B_3) indicated by the hose model and reserving the bandwidth B_3 on link L_3 is wasteful.

- **Virtual Oversubscribed Cluster (VOC):** This model was proposed in [2] as a finer grained version of the hose model. In the VOC model, VMs are organized into clusters and have a hose model guarantee inside each cluster. Clusters are then connected together with per-cluster hoses. The capacity of the per-cluster hose is $B \cdot S/O$, where B is the guarantee of each VM inside the cluster, S is the size of the cluster (number of VMs) and O is the oversubscription factor [2]. Again, to better suit applications, we consider a generalized VOC model that accommodates different guarantees, sizes and oversubscription factors for each cluster, unlike the homogeneous model in [2] that is much more constraining for applications.

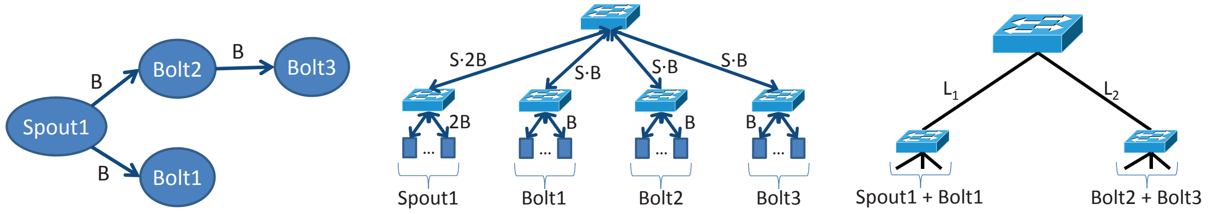
The VOC model is also not well suited to represent most applications. Consider the Storm based application of Fig. 2(a), where for the sake of simplicity, we assume that each component consists of the same number of VMs S, and the outgoing bandwidth of each VM to a communicating component is B. Even for this simple example, there can be many possible VOC model representations, and it is non-trivial to derive a good

[3] Storm components use java threads rather than VMs. The TAG model and our placement algorithm can also be applied to such scenarios, *e.g.,* for Platform-as-a-Service (PaaS) providers.



(a) Simple application (b) Hose modeling (c) Physical deployment example

Figure 1: Three tiered application example deployed using the hose model.



(a) Storm example (b) VOC Modeling (c) Physical deployment example

Figure 2: Storm [4] application example deployed using the VOC model.

one. Fig. 2(b) presents one possible mapping, where we simply represent each application component as a VOC cluster. The resulting model is a degenerate (or relaxed) VOC model, since the clusters are not oversubscribed. Moreover, the model does not accurately capture the communication pattern of the application, since the components do not communicate internally using that bandwidth. The goal behind the VOC model is to isolate higher connected application tiers and place them in better connected topology sub-trees. Having clusters that are not oversubscribed and that do not communicate between their VMs defeats the purpose of the VOC model, and, in fact, has an adverse effect. The placement algorithm may try to place the VMs of each component in separate sub-trees, as Oktopus [2] does, although these VMs communicate only inter-component rather than intra-component, wasting core bandwidth.

Fig. 2(c) shows a potential deployment where two components are placed in one branch of the physical tree while the other two are in a different branch. In this case, the bandwidth reservation on links L_1 and L_2 should be $S \cdot B$ given the communication pattern (since only "Spout1" communicates with "Bolt2" between the two branches). However, VOC will reserve twice this bandwidth (since VOC reserves $\min(3S \cdot B, 2S \cdot B) = 2S \cdot B$). Thus, it is easy to see from this example how VOC can be very inefficient in capturing the bandwidth guarantees. The hose model would be similarly wasteful for this application.

While existing virtual cluster abstractions for expressing guarantees do not efficiently capture even the traffic pattern of our simple examples, real applications can be far more complex. Applications can comprise tens of distinct tiers [13] and use middleboxes (offered by the provider, or implemented through VMs, by companies like F5 or Vyatta). These requirements further strain the virtual cluster models (as we show in §5 for a real trace of datacenter applications).

• **Pipe Model:** An alternative to virtual clusters is to specify pipe guarantees between pairs of VMs [7, 14]. While this model has the ability to capture exactly the traffic needs of the application at a given point in time, it has two major drawbacks. First, it is too rigid and lacks statistical multiplexing. Typically, the VMs belonging to different tiers that exchange data are selected by run-time load balancers, which do not guarantee perfect uniform load distribution to every destination. Thus, even when the aggregate load is constant, the load to each destination varies over time. Users can not update each pipe every minute or second to tightly track the time-varying bandwidth demand of each pipe and will most likely reserve the worst case bandwidth of the peak load for each pipe [6]. For example, a benchmark report on Amazon Elastic Load Balancer shows the sum of the peak loads to each destination is at least double the peak aggregate traffic [15]; thus, the pipe model would lead to a 2X overprovisioning of the bandwidth.

Second, the pipe model is tedious to use. The tenant's request can have hundreds and even thousands of VMs, which, in turn, could result in tens of thousands of pairwise guarantees. Due to this complexity, placing the tenant VMs can also take a long time (§5).

3 Tenant Application Graph (TAG)

We propose the Tenant Application Graph (TAG), a new model that tenants can use to describe bandwidth requirements for applications. Unlike conventional hose

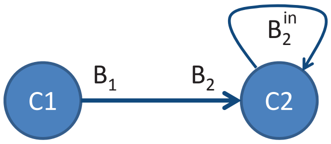


Figure 3: TAG model example.

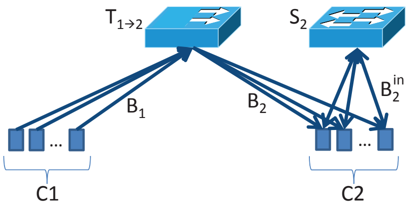


Figure 4: TAG model example explained.

and VOC abstractions, which model physical networks, the TAG abstraction aims to model the actual communication patterns of applications. The TAG model leverages the tenant's knowledge of an application's structure to yield a concise yet flexible representation of the application's communication pattern.

A TAG model is *a graph, where each vertex represents an application component* (or *tier*, we use the two terms interchangeably to indicate the set of VMs performing the same function). Since most applications are conceptually composed of multiple tiers [13], a tenant can simply map each of these tiers onto a TAG vertex. For example, for the application in Fig. 1(a) the tenant would identify three tiers: web, business logic, and database. A special component is used to model all nodes situated outside the datacenter (the Internet).

Tenants request bandwidth guarantees between tiers by placing directed edges between the corresponding vertices in the TAG model. Each directed edge $e = (u, v)$ from tier u to tier v is labeled with an ordered pair $< S_e, R_e >$ that represents per-VM bandwidth guarantees for the traffic. Specifically, each VM in tier u is guaranteed bandwidth S_e for sending traffic to VMs in tier v, and each VM in tier v is guaranteed bandwidth R_e to receive traffic from VMs in tier u.

Having two values (sending and receiving) instead of a single bandwidth guarantee for each edge is useful when the size (*i.e.*, number of VMs) of the two tiers is different. In this way, the total bandwidth outgoing from one tier and incoming to the other tier can be equalized (such that bandwidth is not wasted). If tiers u and v have sizes N_u and N_v, respectively, then the total bandwidth guarantee that the tenant achieves for traffic sent from tier u to tier v is $B_{u\to v} = \min(S_e \cdot N_u, R_e \cdot N_v)$.

To model communication among VMs within tier u, the TAG model allows a self-loop edge of the form $e = (u, u)$ that is labeled with a single bandwidth guarantee $< SR_e >$. In this case, SR_e represents both the sending and the receiving guarantee of one VM in that tier (or vertex). A self-loop edge is equivalent to a conventional hose model, *i.e.*, each VM in tier u can be considered to be attached to a virtual switch via a transmission hose of rate SR_e and a receive hose of rate SR_e.

Fig. 3 shows a TAG model for a simple example application with two tiers $C1$ and $C2$. In this example, a directed edge from $C1$ to $C2$ is labeled $< B_1, B_2 >$. Thus, each VM in $C1$ is guaranteed to be able to send at rate B_1 to the set of VMs in $C2$. Similarly, each VM in $C2$ is guaranteed to be able to receive at rate B_2 from the set of VMs in $C1$.

To better understand the TAG model, Fig. 4 shows an alternative way of visualizing the guarantees expressed in Fig. 3. To model the guarantee between $C1$ and $C2$, each VM in $C1$ is connected to a *virtual trunk* $T_{1\to2}$ by a dedicated directional link of capacity B_1. Similarly, virtual trunk $T_{1\to2}$ is connected through a directional link of capacity B_2 to each VM in $C2$. Thus, an inter-tier edge can be seen as a directional hose model between the VMs of the two tiers.

The TAG model example in Fig. 3 has a self edge for tier $C2$, describing the bandwidth guarantees for traffic where both source and destination are in $C2$ (e.g., the traffic between database servers in Fig. 1(a)). A self edge is equivalent to a hose model between the VMs of that tier. For example, in Fig. 4, each VM in $C2$ is connected through a bidirectional link of capacity B_2^{in} to a virtual switch S_2.

Benefits: the TAG abstraction is intuitive, descriptive and easy to use. Moreover, since the guarantees specified in the TAG model are from any VM of one tier to any set of VMs of another tier, the TAG model naturally accommodates dynamic load balancing between tiers and dynamic re-sizing of tiers (known as "flexing" [12]); *per-VM bandwidth guarantees S_e and R_e do not need to change while tier sizes change by flexing.* This is unlike a pipe model between VMs [7, 14] where *per-pipe* bandwidth guarantees need to be recomputed while load-balancing and flexing, or otherwise the bandwidth must be heavily overprovisioned. Furthermore, with TAG tenants need to specify much smaller number of values than with the pipe model.[4]

We also note that the directional edge definition of TAG naturally accommodates middleboxes, which often impact only one direction of the flow. For example, many load balancers and security services examine only queries to servers but not the reverse traffic from servers; moreover, queries often consume significantly

[4] To be even simpler for users, two edges in opposite directions between two tiers can be combined into a single undirected edge when the incoming/outgoing values for each tiers are the same (*i.e.*, $S_{(u,v)} = R_{(v,u)}$ and $R_{(u,v)} = S_{(v,u)}$).

less bandwidth than responses. The ability to specify directional bandwidths allows TAG to accommodate up to 2× more guarantees than a unidirectional model. For example VM X with a high outgoing guarantee request can be collocated on the same server with VM Y with a high incoming guarantee request.

Model generation: tenants can identify the per VM guarantees to use in the TAG model through measurements [13] or compute them using the processing capacity of the VMs and a workload model. Also, PaaS cloud providers can offer pre-defined service VMs (or JVMs) with pre-computed guarantee requirements.

Previous graph models: we note that graph models are widely used to model components and their connectivity in a variety of fields. One of the closest modeling to our field is in the area of Service-Oriented Computing, in which graph models are used to specify how web or cloud services can be composed from multiple service components with QoS specification [16, 17]. However, none of them tried to capture the bandwidth demand between VMs of a pair of communicating service components and to provide the bandwidth guarantee.

4 TAG Deployment

Deploying the TAG model requires (a) optimizing the placement of VMs on physical servers while reserving needed bandwidth on physical links and (b) enforcing the reserved bandwidths. In this paper, we present a sketch of the placement algorithm and rely on prior proposals (and on-going work) for enforcement [2, 9, 10].

The CloudMirror placement algorithm (CM) tries to maximize tenant placements in a tree-shaped physical topology while meeting all guarantees. For each tenant, CM strives to minimize bandwidth usage of the (oversubscribed) network core by identifying the smallest subtree of the physical topology that may accommodate all VMs of the tenant (similar to Oktopus [2]).

To place VMs in the chosen subtree, CM and Oktopus take different approaches. Oktopus placement relies on the main assumption of the VOC model, that the bandwidth needed within each cluster exceeds the bandwidth needed for inter-cluster communication. Thus, Oktopus places VMs of the same cluster close together in the topology, and places different clusters independently.

In contrast, CM leverages the explicit specification in the TAG model of the bandwidth requirements within and between components to find more efficient VM placements than Oktopus in most cases. CM solves a problem similar to the classic min-cut problem to identify components that are connected in the TAG model with high bandwidth edges (including self-loop edges), and then places the VMs of these heavily communicating components under the same child of the subtree to reduce bandwidth usage at the root of the subtree. The complexity of this min-cut phase is $O(c^2)$ where c is the number of components in a TAG.

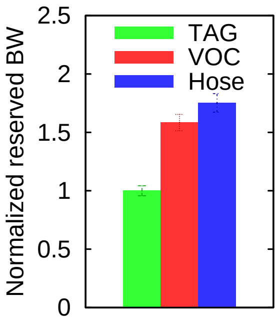


Figure 5: Total bandwidth usage, normalized by that of TAG.

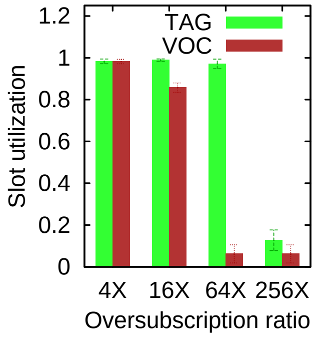


Figure 6: VM slot utilization vs. network core oversubscription ratio.

Components that are too large to fit under a single child must be spread across multiple child nodes, and their bandwidth consumption at the subtree root is often less sensitive to their precise placement within the subtree. For such components, CM shifts to the goal of maximizing server consolidation, by fully utilizing both link bandwidth and other resources (# of VM slots) under individual child nodes: *e.g.*, by placing VMs from a high bandwidth component together with VMs from a low bandwidth component. This is accomplished by solving a problem similar to the classic Knapsack problem; its asymptotic complexity is $O(n)$, n is the number of VMs in a TAG.

The algorithm recursively repeats the two phases (min-cut and Knapsack) for each child sub-tree under the sub-tree chosen for the entire tenant at the beginning of the algorithm, until all the VMs of the tenant are deployed without violating link capacity limits.

5 Evaluation

We use simulation to quantify the efficiency of the proposed TAG model and CM placement algorithm compared to the hose and VOC models placed using Oktopus [2]. We leave assessing the ease of use of the TAG model compared to VOC and pipe models to future work.

We test our algorithms using a workload derived from Microsoft's `bing.com` datacenter, provided by the authors of [18]. Microsoft's workload is formed by a set of services, the service size ranging from one to a few hundred VMs. We consider a tenant job as a set of services that only communicate between themselves (*i.e.*, a connected component in the communication graph containing all services). (As the authors of [18] did, we ignore the management services, to which all other services communicate with.) These connected components exhibit various shapes (*e.g.,* star, linear, mesh), and some services have large intra-service demands (similar to MapReduce) [18]. We assume tenants are sampled uniformly from this distribution in a randomized order. We consider each service as corresponding to a component in the TAG model and to a cluster in the VOC model.

We simulate a tree-shaped 3-level network topology inspired by a real datacenter, with 1024 servers. For simplicity, we assume all VMs have identical CPU and memory requirements, and each server can host 50 VMs.

Fig. 5 compares the total bandwidth reserved throughout the network for the TAG model deployed by the CM placement algorithm to the hose and VOC models deployed by the Oktopus [2] placement algorithm; in fact we added a substantial improvement to the algorithm presented in [2], not described for brevity. We use reserved bandwidth as a comparison metric in order to decouple the efficiency of the models/placements from the capacity constraints of the physical network. For this purpose, we simply assume the network is never a bottleneck and ignore link capacity constraints when placing VMs. In this way, all models and placement algorithms are able to deploy the same set of tenants. We evaluate the impact for different network capacities shortly (in Fig. 6).

Fig. 5 demonstrates the benefits of the TAG model and CM placement algorithm. Oktopus+VOC requires 60% more network resources than CM+TAG, for the same set of tenants (saturating all host VM slots). We have also experimented with synthetic workloads, formed by artificially mixing different application types (*e.g.,* three tier web services and MapReduce jobs), and we obtained similar or better results (2× increase instead of 60%, results omitted for brevity).

In Fig. 6 we consider the network capacity constraint, and evaluate the impact for different network capacities on VM slot utilization. We assume that each link between server and ToR switch supports 10Gbps while capacities of other links are adjusted in order to control oversubscription ratio. We deploy tenants one by one, and we plot the utilization of the datacenter VM slots reached when the first tenant is rejected, *i.e.,* tenant cannot be deployed due to insufficient available bandwidth or CPU/memory resources. We can see that for some oversubscribed networks, CM+TAG can deploy 15× more VMs than Oktopus+VOC before rejecting the first tenant.[5]

The runtime of CM is typically below 100 msec for tenants of up to 100s of VMs and less than 1 second for tenants up to 1000 VMs. This result is promising if we compare it to runtimes of more than 10 mins reported for placing pipe models (for a 1024 VM tenant and a topology with 1024 VM slots) [14]. CM and Oktopus have similar runtimes.

6 Future Work

We are currently collecting and analyzing traffic traces from production datacenters to verify the TAG model and placement algorithm with more real scenarios, for example, tenant churn and flexing (auto-scaling). We plan for OpenStack based implementation and integration with ElasticSwitch bandwidth guaranteeing system [10]. Other future work opportunities include: developing pricing models that benefit both cloud users and operators and extending TAG and its placement algorithm to support anti-affinity rules for resilience to host and rack failures.

Acknowledgments

We thank Peter Bodik and the other authors of [18] for providing us with the data we used in the presented experiments. We also thank Ramana R. Kompella for his input on the earlier stages of the work.

References

[1] "Amazon EC2 Instances - Cluster Networking." `http://aws.amazon.com/ec2/instance-types/`.

[2] H. Ballani *et al.*, "Towards Predictable Datacenter Networks," in *ACM SIGCOMM*, 2011.

[3] "Amazon - Every 100ms delay costs 1% of sales." `http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt`.

[4] "Storm: Distributed and fault-tolerant realtime computation." `http://storm-project.net/`.

[5] D. Lynn, "Storm: The Real-Time Layer Your Big Data's Been Missing," in *Glue Conference*, 2012.

[6] N. G. Duffield, P. Goyal, A. G. Greenberg, *et al.*, "A flexible model for resource management in virtual private networks," in *ACM SIGCOMM*, 1999.

[7] T. Benson *et al.*, "Cloudnaas: A cloud networking platform for enterprise applications," in *ACM Symposium on Cloud Computing (SOCC)*, 2011.

[8] H. Rodrigues *et al.*, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *USENIX WIOV*, 2011.

[9] L. Popa *et al.*, "FairCloud: Sharing the Network in Cloud Computing," in *ACM SIGCOMM*, 2012.

[10] L. Popa *et al.*, "ElasticSwitch: Practical and Efficient Bandwidth Guarantees," in *ACM SIGCOMM*, 2013.

[11] V. Jeyakumar *et al.*, "EyeQ: Practical Network Performance Isolation at the Edge," in *USENIX NSDI*, 2013.

[12] "Amazon Web Services - Auto Scaling." `http://aws.amazon.com/autoscaling/`.

[13] M. Hajjat *et al.*, "Cloudward bound: planning for beneficial migration of enterprise applications to the cloud," in *ACM SIGCOMM*, 2010.

[14] X. Meng *et al.*, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM*, 2010.

[15] Brian Adler, "Load Balancing in the Cloud: Tools, Tips, and Techniques." RightScale Technical Whitepaper.

[16] A. Klein *et al.*, "Towards network-aware service composition in the cloud," in *ACM WWW*, 2012.

[17] Z. Ye *et al.*, "Genetic algorithm based qos-aware service compositions in cloud computing," *LNCS*, vol. 6588, pp. 321–334, 2011.

[18] P. Bodik *et al.*, "Surviving Failures in Bandwidth-Constrained Datacenters," in *ACM SIGCOMM*, 2012.

[5] The bandwidth values in Microsoft's workload data are relative not absolute. We scale the bandwidth values in a way such that both TAG and VOC achieve 100% slot utilization when the network is not oversubscribed. Note that the bandwidth scaling did not affect Fig. 5 as we ignored network capacity constraints and normalized the bandwidth consumptions in Fig. 5.