The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers

Di Xie, Purdue University
Ning Ding, Purdue University
Y. Charlie Hu, Purdue University
Ramana Kompella, Purdue University

ABSTRACT

In multi-tenant datacenters, jobs of different tenants compete for the shared datacenter network and can suffer poor performance and high cost from varying, unpredictable network performance. Recently, several virtual network abstractions have been proposed to provide explicit APIs for tenant jobs to specify and reserve virtual clusters (VC) with both explicit VMs and required network bandwidth between the VMs. However, all of the existing proposals reserve a fixed bandwidth throughout the entire execution of a job.

In the paper, we first profile the traffic patterns of several popular cloud applications, and find that they generate substantial traffic during only 30%-60% of the entire execution, suggesting existing simple VC models waste precious networking resources. We then propose a fine-grained virtual network abstraction, Time-Interleaved Virtual Clusters (TIVC), that models the time-varying nature of the networking requirement of cloud applications. To demonstrate the effectiveness of TIVC, we develop PROTEUS, a system that implements the new abstraction. Using large-scale simulations of cloud application workloads and prototype implementation running actual cloud applications, we show the new abstraction significantly increases the utilization of the entire datacenter and reduces the cost to the tenants, compared to previous fixed-bandwidth abstractions.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations

General Terms: Algorithms, Design, Performance

Keywords: Datacenter, Network Reservation, Allocation, Bandwidth, Profiling

1. INTRODUCTION

Cloud computing has transformed the enterprise computing landscape significantly. By offering virtually unlimited resources without any upfront capital investment and a simple pay-as-you-go charging model, cloud computing provides a compelling alternative to enterprises constructing and maintaining their own cluster-computing infrastructure. The long-term viability of cloud computing depends, among others, on two major factors—cost and performance. Unless cloud platforms prove cheaper than enterprise ones, enterprises have no incentive to migrate their computational requirement to the cloud. Similarly, cloud platforms need to provide some guarantees about job performance, otherwise enterprises may be apprehensive to migrate to the cloud to begin with.

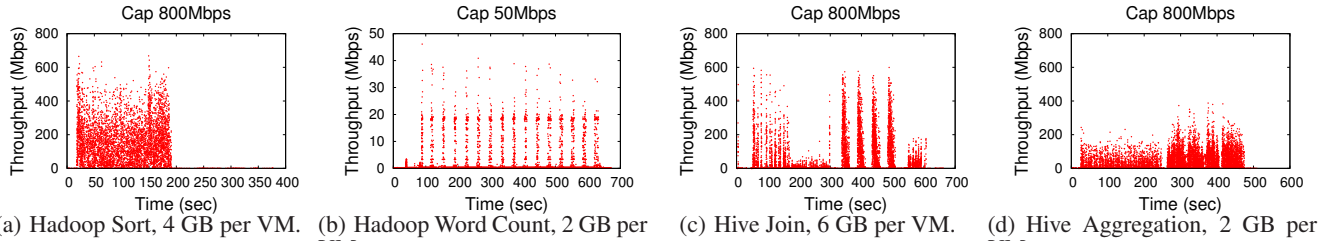
Unfortunately, today's public cloud platforms such as Amazon EC2 do not provide any performance guarantee, which in turn affects tenant cost. Specifically, the resource reservation model in today's clouds only provisions CPU and memory resources but ignores networking completely. Because of the largely oversubscribed nature of today's datacenter networks (*e.g.,* [19]), network bandwidth is a scarce resource shared across many tenants. When networking intensive phases of applications collide and compete for the scarce network resources, their running times become unpredictable. The uncertainty in execution time further translates into unpredictable cost as tenants need to pay for the reserved virtual machines (VMs) for the entire duration of their jobs.

Recent works such as SecondNet [20] and Oktopus [11] noted this lack of networking SLA in cloud environments and proposed new network abstractions for tenants to specify their networking needs along with their CPU and memory needs, so that their applications can obtain predictable performance. For example, SecondNet proposes bandwidth reservation between every pair of VMs. Oktopus proposes a simpler virtual cluster (VC) model where all VMs are connected to a virtual switch with links of bandwidth B. While both models provide a good start, they fail to capture the *temporal dimension* of network resource requirement. Specifically, we observe that the networking requirement of many applications experience significant changes throughout their execution. Hence provisioning a single bandwidth B for an entire cluster or for each pair of VMs throughout the job execution is clearly wasteful.

To illustrate this, we show the instantaneous network throughput of the MapReduce Sort application in Figure 1(a) (details are in §2.2). We observe from the figure that the Sort application utilizes the network only during the first half of the execution (during the shuffle phase), while the second half (merge sort and reduce phase) requires very little networking resource. Effectively the network utilization is in the form of a simple square wave. Figure 1 further shows the network utilization characteristics of several other applications such as Word Count, Hive Join and Hive Aggregation. While they exhibit more complicated network characteristics than Sort, a common takeaway is the relatively sparse, time-varying networking requirement.

Using a fixed bandwidth reservation can potentially waste precious resources of the datacenter. Consider several Sort jobs that have bandwidth requirement B during the first half of their execution. If we use Oktopus' virtual cluster (VC) model for specifying the network requirement of Sort, the fixed bandwidth B is provisioned during the entire duration of the job, preventing another Sort

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.



(a) Hadoop Sort, 4 GB per VM. (b) Hadoop Word Count, 2 GB per VM. (c) Hive Join, 6 GB per VM. (d) Hive Aggregation, 2 GB per VM.

Figure 1: Time-varying traffic demand of cloud applications.

job from using the unused bandwidth B during the second half of the job. Clearly, if there are enough VMs, we can schedule a second Sort job such that the first half of the second job overlaps with the second half of the already running first job.

Our main contribution in this paper is to exploit this key observation and explore a new, fine-grained network reservation abstraction called *temporally-interleaved virtual clusters* (TIVC), that allows specifying the time-varying networking requirement of cloud applications. By capturing the temporal networking usage, the TIVC abstraction reduces over-reservation of network bandwidth in fixed-bandwidth abstractions, allowing more jobs to be admitted into the datacenter. Thus the new abstraction not only improves the utilization of the network resource but also potentially increases the VM utilization and hence the overall datacenter utilization. The increased utilization of the entire datacenter in turn translates into either increased cloud provider revenue, *e.g.,* under today's VM-time only charging model, or both increased provider revenue and reduced tenant cost, when the cloud provider adopts a bandwidth charging model (*e.g.,* [30, 10]), compared to fixed-bandwidth abstractions.

For the TIVC abstraction to be viable in practice, there are two fundamental questions that we must answer: How can we obtain the TIVC model corresponding to a job? How do we allocate and provision jobs with TIVC specifications in a physical datacenter? While a similar question to the first confounded prior network abstractions [11, 20], they simply assumed that the customer will specify them somehow. In this paper, we propose a systematic, profiling-based methodology for automatically generating TIVC model parameters for a given application. There are several challenges that we need to tackle to make this profiling-based methodology practical.

The first challenge concerns the elastic nature of networking requirement for many jobs; as we provide more bandwidth, job will finish faster. So how should one determine the peak bandwidth requirement for an application in deriving its TIVC model, *e.g.,* in profiling the application? Using a detailed study using real cloud applications, we provide insights into this question, which previous works have ignored completely.

The second challenge concerns designing TIVC model functions so that they can be easily generated, they capture the time-varying networking requirement of an application well, and yet they can be practically enforced in the datacenter network. We propose to convert the networking requirement output of the profiling stage into coarse-grained pulse functions (square waves) with different widths and heights. Our conversion scheme ensures the derived bandwidth reservation model has little impact on the job completion time.

The final challenge concerns a practical allocation algorithm that not only needs to identify where a TIVC job can be admitted spatially (in the network topology), but also temporally (when the bandwidth will be freed in the future to accommodate this job throughout its execution). We propose an allocation algorithm based on dynamic programming that is highly scalable and finds valid allocations with the best spatial locality, *i.e.,* in the lowest subtree available.

We have developed PROTEUS, a system that implements the TIVC abstraction. PROTEUS derives TIVC models for applications, accepts jobs online and allocates them to the physical datacenter, and provisions network bandwidth according to the TIVC models. We deployed our prototype on an 18-node datacenter with a ternary tree topology using NetFPGA-based switches. Our simulations of a large-scale datacenter using real MapReduce application workload show our TIVC abstraction can increase the datacenter batched job throughput by 34.5%, and reduce the rejection rate of dynamically arrived jobs from 9.5% to 3.4%, which translates into 22% higher cloud provider revenue than the fixed-bandwidth abstraction under today's VM-time based charging model. When the cloud provider moves to a bandwidth charging model, TIVC can both improve cloud provider revenue by 11% and reduce tenant cost by 12%, compared to the fixed-bandwidth abstraction.

2. BACKGROUND AND MOTIVATION

In this section, we first review the state of the art, and then motivate the time-varying networking abstraction proposed in this paper.

2.1 State of the Art

To provide performance guarantees, several recent works [20, 11] have proposed *virtual network cluster* abstractions, which allow cloud users to specify not only the type and number of VMs requested, but also the associated networking requirement, *i.e.,* the bandwidth requirement between the VMs. Such a virtual cluster gives cloud users an assurance of their job performance based on the VM and networking SLAs.

SecondNet [20] proposes APIs that let users specify either end-to-end bandwidth for each pair of VMs (suitable for strong guarantees), or ingress/egress bandwidth for each VM (for better than best-effort sharing). Oktopus [11] proposes two simplified abstractions: virtual clusters and virtual oversubscribed clusters. A virtual cluster is a one-level logical tree topology, specified as $<N, B>$, which asks for N VMs and each VM is connected to a virtual switch by a bidirectional link of bandwidth B. A virtual oversubscribed cluster request, $<N, B, S, O>$, specifies a two-level logical tree topology with an oversubscription ratio of O.

All the models above, however, only allow *fixed* bandwidth guarantees, since they fundamentally assume the applications have the same bandwidth requirement throughout the entire execution, which is rarely true in practice. Specifically, our *key observation* is that typical cloud applications generate varying amount of traffic in different phases of their execution. Thus reserving their peak bandwidth requirement for the entire execution wastes scarce network resources in the datacenter, which reduces the number of jobs that can fit in the datacenter to run concurrently, and hence the overall utilization of the datacenter.

2.2 Profiling Networking Demand

To assess the extent of networking requirement variations of cloud applications, we conduct a measurement study. Since MapReduce [17] is a popular programming paradigm for large-scale data-intensive operations and many cloud applications have been ported to MapReduce, such as search indexing, database, and machine learning, we profile the networking patterns of several typical MapReduce jobs, including Sort, Word Count, Hive Join, and Hive Aggregation, which have been used as the primary benchmarks in recent datacenter studies (*e.g.,* [16, 36, 24, 28]). While this list of applications is not exhaustive, we believe they represent an important class of applications found in datacenters.

Methodology. We conducted the profiling experiments on our 33-machine cluster, using open-source Hadoop 0.20.0 as the MapReduce platform. The 33 machines are connected to a Gigabit switch. Each of the 33 machines is equipped with a 4-core Xeon 2.4GHz processor, 4GB memory, and running 2 Xen virtual machines (domU) and each VM is allocated 1.5GB memory and a dedicated hard drive. To obtain the network traffic, we deployed tcpdump at all the Hadoop slave VMs to log all communication (packet headers) between them. Since the bidirectional traffic are almost identical for the set of applications, for each application, we plot the access egress bandwidth of one VM in each time bin of 10ms, which is in the same order of RTTs under load and hence sufficiently small to capture the transient peak throughput. The throughput for different VMs are very similar, and we study the variation in Section 4.

Traffic Patterns of Cloud Applications. We observe that the traffic patterns of the set of applications fall into the following three categories, of increasing generality. In all categories, a base bandwidth needs to be reserved to facilitate job maintenance tasks such as communicating with the job scheduler, and to support low volume network traffic among the VMs.

Type 1: Single peak. The Hadoop *Sort* benchmark sorts randomly generated records with 10-byte keys and 90-byte values. Figure 1(a) plots the network throughput over the execution with 4 GB input data per VM. We observe that most of the network traffic were produced in the first half of the job execution. In other words, the traffic pattern exhibits a single peak, defined as a continuous period of throughput above some base amount.

Type 2: Repeated fixed-width peaks. The Hadoop *Word Count* benchmark counts the number of word occurrences in the input data. Figure 1(b) shows that there was only a small amount of data been shuffled over the network periodically throughout the execution. The small amount of data is due to reduced map output from enabling the combiner in map tasks, and the periodic network traffic is because the map tasks finished in batches, and hence their output were shuffled periodically. Thus the traffic demand exhibits repeated bursts with a similar amount of traffic volume per burst, and consequently similar duration.

Type 3: Varying-width peaks and *Type 4: Varying height and width peaks. Hive* [2] implements a data warehouse system built on top of Hadoop. It provides an SQL-like language for data queries. The Hive performance benchmark [5] is used to compare the performance of Hadoop, Hive, and PIG [3]. It consists of four queries: grepping from a table, selecting columns from a table, aggregating a table, and joining two tables. We generated a 4GB UserVisits table and a 2GB Rankings table per node. Figures 1(c)-(d) plot the throughput for the two queries Join and Aggregation. We observe that Hive Join exhibits a brief burst of network activity at the beginning, followed by a longer duration of less-intensive traffic from map tasks, followed by five bursts, corresponding to five reduce tasks. Hive Aggregation exhibits a long duration of moderate traffic demand, followed by a long duration of more intense traffic demand. These two applications fall into the more general category of a sequence of traffic bursts of varying durations and of varying intensities.

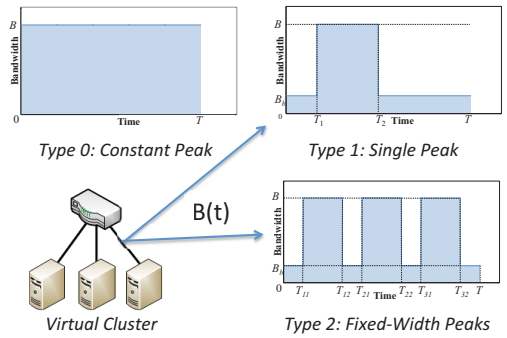


Figure 2: TIVC models contain time-varying bandwidth specifications $B(t)$.

3. THE TIVC ABSTRACTION

Our key observations in the previous section clearly indicate the need for a new networking abstraction that can express the time-varying nature of application networking requirement. To this end, we propose a novel network abstraction called *temporally-interleaved virtual cluster (TIVC)* that captures the temporal variations in the network behavior of cloud applications.

The TIVC abstraction (shown in Figure 2) consists of a virtual cluster of N nodes connected to a switch, via links of bandwidth B, similar to the VC abstraction proposed in [11]. However, the key difference is that the bandwidth for each link is a time-varying function $B(t)$ instead of a constant value in prior work. This allows capturing the actual networking requirement of applications much more precisely, which enables the cloud provider to achieve better utilization of datacenter resources, and ultimately improves the provider revenue and reduces the tenant cost without sacrificing the job performance, as we will show in our experiments.

3.1 Different TIVC Models

Given the networking requirement profile of an application, *e.g.,* shown in Figure 1, one can potentially derive some complicated function (*e.g.,* high-order polynomials) to precisely model the changing requirement over time. However, such smooth functions significantly complicate the process of allocating a TIVC in the physical datacenter network, as well as provisioning of the continuously changing bandwidth requirement in the physical network. To strike a balance between modeling precision and implementation difficulties[1], we choose to model the networking requirement as simple pulse functions in this paper. We leave exploring other tradeoffs in the spectrum as future work.

Since TIVC is a generalization of the VC abstraction, we call the model with a fixed bandwidth $B(t) = B$ as Type 0 as shown in Figure 2. To capture the several general time-varying patterns observed in our profiling study, we propose the following model functions, also shown in Figure 2.

Type 1: Single peak. A Type 1 model captures the networking demand of applications that only generate network traffic in a certain interval, and has a format of <N, T, B_b, P>, where P=(T_1, T_2, B). The bandwidth function is given as

$$B(t) = \begin{cases} B_b & : t \in [0, T_1] \text{ or } [T_2, T] \\ B & : t \in [T_1, T_2] \end{cases}$$

[1]The fixed bandwidth specification in [20, 11] can be viewed as going to one extreme in this tradeoff, ease of implementation.

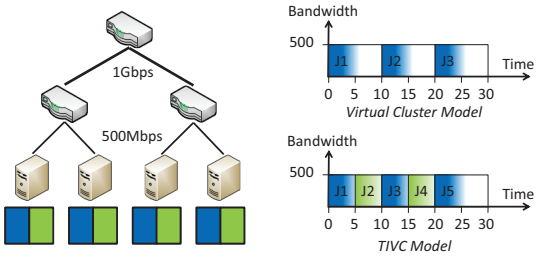


Figure 3: Simple tree with full-bisection bandwidth.

For example, the Sort application in Figure 1(a) would request for a Type 1 TIVC with <N, $382s$, $4Mbps$, ($18s$, $191s$, $800Mbps$)>.

Type 2: Fixed-width peaks. The Type 2 TIVC model captures the networking requirement of applications that have repeated traffic peaks. A request of this type has the format <N, T, B_b, P_1, ..., P_K>[2], where $P_i = (T_{i1}, T_{i2}, B)$. The bandwidth function is

$$B(t) = \begin{cases} B & : t \in [T_{i1}, T_{i2}], i \in [1, K] \\ B_b & : otherwise \end{cases}$$

The request specifies K repeated peaks of bandwidth B and the same width to be provisioned, *i.e.,* the widths of the peaks ($T_{i2} - T_{i1}$) are the same, and the base bandwidth B_b during the rest of time. For example, the Word Count application in Figure 1(b) would request for a Type 2 TIVC with <N, $672s$, $2Mbps$, ($80s$, $92s$, $50Mbps$), ..., ($620s$, $632s$, $50Mbps$)>.

Type 3: Varying-width peaks. The Type 3 TIVC model is more general than previous two types and captures the networking requirement of applications that have varying-width traffic peaks. The request format and the bandwidth function are the same as with Type 2, except the durations of the peaks ($T_{i2} - T_{i1}$) can differ. For example, Hive Join in Figure 1(c) would request for a Type 3 TIVC with <N, $672s$, $50Mbps$, ($52s$, $172s$, $800Mbps$), ($336s$, $361s$, $800Mbps$), ($387s$, $412s$, $800Mbps$), ($434s$, $459s$, $800Mbps$), ($485s$, $508s$, $800Mbps$), ($550s$, $607s$, $800Mbps$)>.

Type 4: Varying height and width peaks. The Type 4 TIVC model refines the Type 3 model to allow varying heights for different peaks. The format is the same as in Type 3, except $P_i = (T_{i1}, T_{i2}, B_i)$, which specifies a bandwidth cap of B_i is requested from time T_{i1} to T_{i2}.

The four models based on pulse functions are of increasing generality. While we do not claim they are universal, we find they capture well the traffic patterns of the applications we have studied (which are also widely used in previous datacenter networking studies, *e.g.,* [16, 36, 24, 28]).

3.2 Implication on Job Scheduling

By more precisely capturing the networking requirement of applications, TIVC enables the cloud provider to schedule more jobs to run concurrently not only in over-subscribed networks, but also in networks with full-bisection bandwidth, such as fat-trees [7, 19].

Consider the simple tree datacenter network with full-bisection bandwidth shown in Figure 3. Consider a sequence of Sort-like jobs, each requesting 4 VMs and access bandwidth of 500Mbps during the first half of their 10-second execution. Under the VC model which reserves the constant 500Mbps bandwidth throughout the job execution, only one job can be scheduled to run every 10 seconds. The four VMs have to be allocated on the four servers since each server has an access bandwidth of 500Mbps. In contrast, under TIVC, after the first job has run for 5 seconds and thus finished the networking phase, the second job can be scheduled to run on the other VM of each of the four servers. This results in doubling the resource utilization and hence the job throughput of the whole system.

[2]We enumerate the peaks in all types for consistency.

4. TIVC MODEL GENERATION

In this section, we study the key challenge in using TIVC abstractions in practice: How to automatically generate the TIVC model for a given cloud application? Since the quantitative program behavior, *e.g.,* networking requirement, is typically dependent on the MapReduce framework configurations at runtime (*e.g.,* [22]) and potentially on input parameters, in this paper, we propose a "black-box" approach to modeling the traffic requirement of a cloud application. The general idea is to collect the traffic trace of the application during profiling runs,[3] and use it in model generation. We discuss the generality of our approach in §4.5.

Realizing the above profiling-based approach faces two immediate challenges. First, there exists a tradeoff between the bandwidth cap and the execution time, since tightening the bandwidth constraint elongates the networking component of a job and affects the completion time. The question then is what bandwidth cap should be used during the profiling run? Second, given a traffic profile collected from the profiling run, how to automatically derive the most suitable TIVC model.

4.1 Impact of Bandwidth Capping

Under the TIVC abstraction, the cloud provider charges for both the VMs and the network usage[4]. This raises the question of how to balance the VM cost and the networking cost, *e.g.,* in trying to minimize the total cost of a tenant's job. Intuitively, a job can request for a lower bandwidth limit which lowers the networking cost per unit time and potentially the total networking cost, which however may lead to longer networking time and hence longer job execution time, increasing the total VM cost. To our knowledge, this important question has not been studied before, even in immediately relevant prior works such as Oktopus [11]. In the following, we conduct experiments to characterize the first-order impact that bandwidth capping has on the cloud application execution time from which we draw implications to TIVC model generation.

We repeat the profiling experiments in §2.2 while gradually reducing the access bandwidth limit per VM. Figure 4 shows the measured application execution times. We make two observations. First, for each application, until the bandwidth cap is reduced to a certain threshold (*e.g.,* 300 Mbps for Hive Join), there is virtually no impact on the application execution time and network throughput. Second, once the cap crosses below the threshold, denoted as the *no-elongation threshold bandwidth*, the execution time is elongated monotonically. We empirically confirmed that the elongation of the execution time is due to the slowdown of the application networking activities, as we measured the execution time slowdown to be equal to the elongation on the network active periods, *i.e.,* the width of the pulses captured in the TIVC model shown in §4.2.

The main reason for the above behavior is that cloud applications have mixed communication and computation even in network-intensive phases. For example, we calculated the average throughput per VM of Hive Join during each of the four high pulses in Figure 1(c) which had a bandwidth cap of 800 Mbps to be only 160 Mbps. This indicates that it may be unnecessary to set the bandwidth cap to be much higher than the average application traffic generation rate. However, we found that capping the bandwidth to

[3]We note profiling traffic demand is required even in Oktopus [11] to meaningful decide on the constant bandwidth parameter B.

[4]Designing bandwidth charging models is currently being studied [30, 10] and is beyond the scope of this paper.

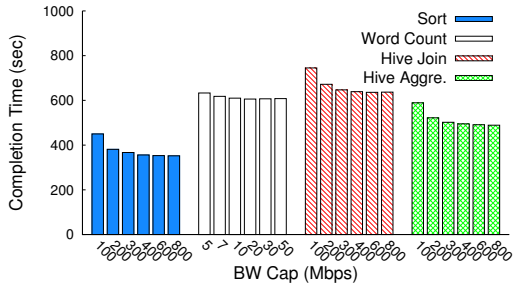


Figure 4: Application execution time under different bandwidth caps.

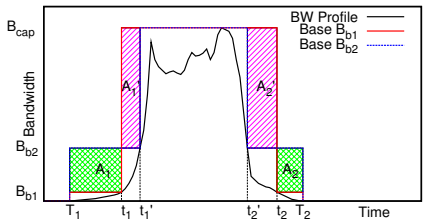


Figure 5: Model fitting by varying the base bandwidth.

be exactly the same as the application data generation rate is too rigid, *i.e.,* it actually slows down the application network-intensive phases. This is shown in Figure 4 where Hive Join under 200 Mbps runs 25 seconds longer than under 300 Mbps. The reason is mainly due to applications' bursty networking behavior. Specifically, we observe that applications tend to interleave computation and data transfer phases, *i.e.,* it generates traffic in bursts during data transfer phases, at a rate higher than the average data generation rate. Further, the computation has certain dependence on the data transfer. For example, in MapReduce jobs, the data shuffling phase is interleaved with map tasks which generate processed data. If the bursty data transfer is slowed down due to bandwidth capping, it pushes back the subsequent computation.

The above threshold behavior suggests that the tenant should always pick the bandwidth cap to be the same or lower than the no-elongation threshold bandwidth, as higher capping wastes networking cost without making the application run faster. However, below the threshold, automatically deriving the bandwidth cap that optimizes the total cost requires a precise modeling of the relationship between the bandwidth cap and application execution time, which is beyond the scope of this paper. Instead, in §5.1, we implement a profiling-based approach to help the user to pick the bandwidth cap that optimizes the total cost while meeting the performance goal.

The above study further helps us to draw an important implication in terms of automatic TIVC model generation. If the user picks a bandwidth cap B_{cap} equal to or below the threshold value in the profiling run to be used for model generation, in the generated TIVC model, the ceiling of the pulses should be conservatively set to be the same as the bandwidth cap, as using a lower ceiling will elongate the application execution time relative to the profile run. However this rule can be loosened for certain pulses which will be become clear in §4.3.

4.2 Model Generation

The model generation algorithm takes as input the traffic profile of an application profiling run under bandwidth cap B_{cap}, and derives the TIVC parameters that achieves the highest efficiency in the following two steps. Here, we define *efficiency* of a TIVC model as the ratio of the total application traffic volume over the total traffic volume under the bandwidth reserved by the TIVC model, *i.e.,* $\int_0^T B(t)dt$.

Since TIVC models use pulse functions, *i.e.,* square curves, the main idea of automatic model generation is to derive square curves to cap the continuous bandwidth demand curve from the profiling run. There are potentially many ways of generating such bounding square curves, and different bounding curves may have different efficiencies. We note high efficiency fittings (and hence low total bandwidth volume) may not necessarily translate into fitting more jobs in the datacenter, as how well the TIVC models of competing jobs complement each other also plays an important role. However, we envision it is more practical that TIVC models are generated offline, *i.e.,* oblivious to the competing jobs at (future) scheduling time, and hence set maximizing the efficiency as the main objective in model generation.

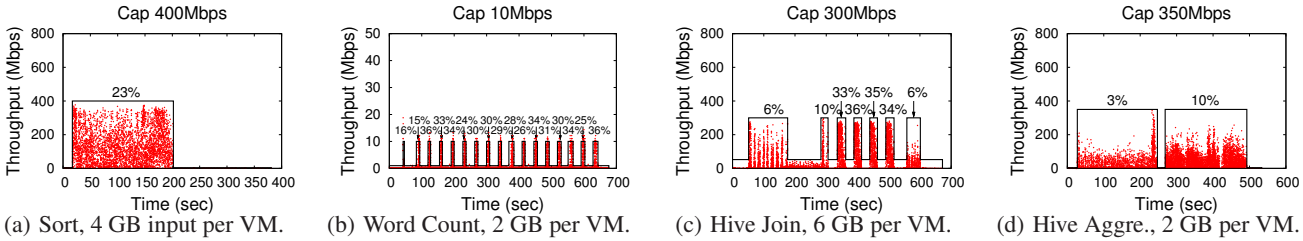
We generate the most efficient Type 4 model in two steps. First we show how to generate the bounding square curves that maximize the efficiency under Type 3 TIVC (which generalizes Type 1 and 2). We then show the conditions when a Type 3 model is refined into Type 4 in §4.3. Recall a Type 3 TIVC model specifies a base bandwidth B_b and a list of fixed-height pulses $P_i = (T_{i1}, T_{i2}, B)$. For a fixed B_b value, to meet the application's bandwidth need, all periods in which the bandwidth usage is observed to be greater than B_b is conservatively rounded up to B_{cap}, for reasons discussed in §4.1. Thus B_b effectively controls the relative amount of bandwidth volume under the base bandwidth periods and under the pulse periods. In general, the higher the B_b value, the more area under the base bandwidth, and the less area under the (narrower) pulses, as shown in Figure 5, which shows two tentative square curve fittings with base bandwidth B_{b1} and B_{b2}, respectively. Now given an application bandwidth profile, the Type 3 TIVC parameters that maximize the model efficiency can be found by searching through different values of B_b.

Figure 6 shows the bandwidth profiles for the four applications shown in Figure 1 but under the no-elongation threshold bandwidth cap, and the corresponding TIVC models generated for these profiles. We make two observations. First, compared to Figure 1, using threshold bandwidth capping did not elongate the application execution, but smoothed the traffic peaks. Second, the TIVC models generated for the four applications are of Types 1, 2, 3 and 3, as expected, and achieve 22.9%, 8.9%, 13.1%, and 6.3% efficiencies, respectively.

4.3 Model Refinement

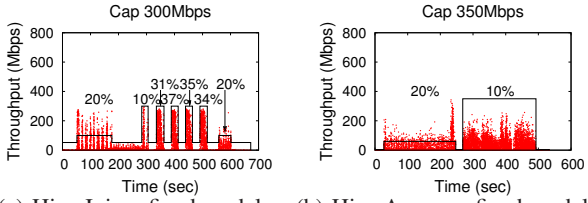
The Type 3 TIVC model generated for a given application traffic profile consists of square curves of two different bandwidth limits, base bandwidth B_b during valleys and capping bandwidth B_{Cap} during peaks. A close look at the individual pulses and valleys in Figure 6 reveals two findings. First. if we calculate the efficiencies of different pulses (shown in Figure 6 next to each pulse), the values can differ significantly, with the peak efficiency being around 30%. Second, the profile for the same application under a lower bandwidth cap (not shown due to page limit) shows the pulses with the highest efficiency get elongated first, and with low efficiencies can sustain a lower bandwidth cap without being elongated.

To improve the efficiency of the TIVC model, which allows the cloud provider to potentially fit more jobs in the datacenter, we refine the generated Type 3 model by lowering the bandwidth cap for pulses that have very low bandwidth efficiencies, using the following heuristic. If the efficiency is lower than a threshold γ, we lower the bandwidth cap so that the efficiency is around α. We empirically found for all the applications we studied, setting $\gamma = 8\%$ and $\alpha = 20\%$ is sufficiently conservative, *i.e.,* it will not elongate the pulses when running under the new bandwidth caps. Figure 7



(a) Sort, 4 GB input per VM. (b) Word Count, 2 GB per VM. (c) Hive Join, 6 GB per VM. (d) Hive Aggre., 2 GB per VM.

Figure 6: Bandwidth profiles under no-elongation threshold bandwidth cap and their Type 3 models.



(a) Hive Join refined model. (b) Hive Aggre. refined model.

Figure 7: Hive Join and Hive Aggre. refined model.

Table 1: TIVC offsets, measured as the standard deviation of the rise/fall timings of pulses across 32 VMs.

	Sort	WC	Hive Join	Hive Aggre.
Pulse 1 (rise, fall)	5.8, 7.9	3.5, 4.0	2.0, 3.2	0.1, 2.3
Pulse 2 (rise, fall)	N/A	5.3, 5.6	8.1, 3.6	0.1, 0.3
Pulse 3 (rise, fall)	N/A	5.4, 5.8	1.9, 2.7	N/A
Pulse 4 (rise, fall)	N/A	5.1, 5.5	1.9, 2.4	N/A
Pulse 5 (rise, fall)	N/A	5.7, 5.4	1.9, 2.4	N/A
Pulse 6 (rise, fall)	N/A	5.5, 5.5	1.9, 2.4	N/A
Pulse 7 (rise, fall)	N/A	5.3, 5.5	2.1, 2.6	N/A

shows the Type 4 models for Hive Join and Hive Aggre. which are refined from their corresponding models in Figure 6. We do not claim the two threshold values to be general and suitable threshold values are likely dependent on the networking and computation mix of the class of applications. Rather, the key point is that Type 4 models can be systematically derived by refining Type 3 models from lowering the bandwidth cap for pulses with low efficiencies.

4.4 Incorporating Model Offsets

A final challenge faced by TIVC model generation is concerned with the TIVC models for different VMs of the same job. There are two related questions: how much do the traffic demand and hence the TIVC models generated for different VMs differ, and if the TIVC models are the same, how well are their timings aligned. Our profiling study shows that for the set of MapReduce jobs, the traffic demand and hence the TIVC models generated across the VMs are of the same type and with the same number of pulses. However, the rising and falling edges of the pulses can be offset, potentially due to the delay in task dispatching from the task scheduler. Table 1 lists the standard deviation of the rise and fall timings of the pulses for the four applications. We see that the standard deviation of the pulses across 32 VMs is less than 9 seconds.

The above small misalignment between TIVCs of different VMs suggests that instead of generating individual TIVCs for the VMs, we can generate a single TIVC which is easier to provision, as follows. We first process per-VM profiles to generate per-VM refined TIVC models. We then calculate the max of their base bandwidths, and regenerate per-VM TIVC models using this new base bandwidth. Next we merge the new per-VM TIVC models by merging the corresponding pulses, taking the max of their widths and heights. Figures 8(a)-(c) show the traffic profiles and TIVCs for three randomly sampled VMs and Figure 8(d) shows the merged TIVC, for Hive Join. Since the threshold bandwidth cap for Word Count, 10Mbps, is already very low, it should be provisioned throughout the application execution. The final TIVC models generated for the rest three applications are shown in Table 2.

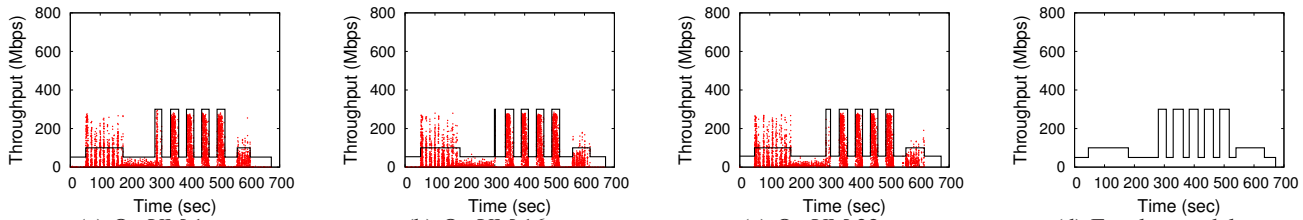
4.5 Discussions

We discuss the generality and limitation of our profiling-based model generation. Our model generation uses the enveloping technique (§4.4) to tolerate small offsets among the traffic demand by different VMs of a job. This works well for the class of MapReduce-type applications which tend to be highly regular in nature – the worker VMs are performing similar tasks and hence are likely to generate traffic of similar volume and at similar times. For applications that generate non-uniform traffic, we can generate and enforce per-VM TIVC models.

Our approach assumes the input data size per VM stays the same during profiling runs and production runs. One potential source of variation between the traffic patterns during profiling runs and production runs is the input data, as the processing time of different data items and hence across the VMs could be uneven. In our experiments with input data generated using random seeds, we did not find much difference in the traffic characteristics across runs. Specifically, the standard deviation of the pulse edge timings across 5 runs of the four applications using input data generated using random seeds is less than 10 seconds (not shown due to page limit). In general, it is important to validate this assumption for any candidate application across multiple sample profile runs before using the TIVC models.

We envision the primary use scenarios of TIVC models are when customers repeatedly run the same type of jobs with the same input size (and hence same number of VMs), with potentially similar data sets from run to run. Such a scenario is common in iterative data processing (*e.g.,* [14]) such as PageRank [29], HITS (Hypertext-Induced Topic Search) [25], recursive relational queries [12], social network analysis, and network traffic analysis where much of the data stay unchanged from iteration to iteration, and is observed in many production environments (*e.g.,* in Bing's production clusters [6]), where the same job needs to be repeated day in and day out, and the data change slightly. In such scenarios, the jobs could be profiled on each run or periodically, with the TIVC models generated to help schedule the cluster during the next run.

Finally, as with all other network reservation approaches (*e.g.,* [20, 11]), TIVC faces a number of uncertainties during job execution: the data processed by each VM (*e.g.,* a map task) may not be on the local disk, a job execution may experience stragglers or task failures which require some tasks to be re-executed (*e.g.,* [9]), or some VMs or network elements may fail. In general, predictable performance in the presence of such uncertainties requires adding fault tolerance to applications and overprovisioning of not only network resources, but also extra VMs, for accommodating backup tasks. We leave dealing with such uncertainties as future work.



(a) On VM 1. (b) On VM 16. (c) On VM 32. (d) Envelop model.

Figure 8: Bandwidth profile and generated Type 4 TIVC models of Hive Join.

Table 2: TIVC models generated for the applications.

App.	Type	TIVC	Efficiency
Sort	1	<N, 382s, 4$Mbps$, (17s, 202s, 400$Mbps$)>	22.8%
Hive Join	4	<N, 672s, 50$Mbps$, (46s, 183s, 100$Mbps$), (284s, 301s, 300$Mbps$), (329s, 363s, 300$Mbps$), (383s, 413s, 300$Mbps$), (434s, 464s, 300$Mbps$), (485s, 517s, 300$Mbps$), (539s, 634s, 100$Mbps$)>	17.6%
Hive Aggre.	4	<N, 535s, 4$Mbps$, (27s, 253s, 60$Mbps$), (268s, 492s, 350$Mbps$)>	10.7%

5. THE PROTEUS SYSTEM

To demonstrate the effectiveness of TIVC models, we have developed a cloud sharing system called PROTEUS that implements the TIVC models.

5.1 Overview

The goal of PROTEUS is to allow cloud customers to obtain predictable performance and cost guarantees for their applications. This is achieved via three steps, as outlined in Figure 9. In the first step, the customer's application is profiled under different configurations, *i.e.,* of input data size per VM and bandwidth cap, and TIVC models are generated for the profiling runs using the techniques presented in §3. We note the profiling overhead can be drastically reduced when customers repeatedly run the same type of jobs with the same input size (see §4.5).

Each profiling run under a configuration in the first step results in a TIVC model and a service completion time. In the second step, the charging model published by the cloud provider is used to estimate the cost for the candidate TIVC models under different configurations. The customer can then pick whichever configuration that best suits her performance/cost objective.

In the final step, given a TIVC job configuration that the customer picks, the cloud provider runs a spatial-temporal TIVC allocation algorithm to place the job in the physical datacenter in a way that maximizes the utilization and hence the revenue of the cloud datacenter, and configures the datacenter network to enforce the requested time-varying bandwidth specified in the TIVC specification. We describe the details of these two components next.

5.2 Spatial-Temporal Allocation

The job manager implements the TIVC allocation algorithm to allocate VM slots on the physical machines in an online fashion. It achieves this by maintaining up-to-date information of (1) the datacenter network topology; (2) the empty VM slots in each physical machine; and (3) the residual bandwidth for each link, calculated from tallying the TIVC allocations of currently running jobs. We focus on tree-like topologies such as multi-rooted tree topologies which are typical of today's datacenters. In such a topology, machines are grouped into racks and the Top-of-Rack (ToR) switches are in turn connected to higher level switches.

We present a generic allocation algorithm for all TIVC models, each of which can be viewed as a sequence of pulses of different bandwidth and duration. We first show how to find a valid allocation for a TIVC request, then show how to find a good allocation out of all the valid allocations. Our allocation algorithm improves the Oktopus allocation algorithm [11] with a novel dynamic programming solution in searching valid allocations, which not only significantly improves the search efficiency, but also guarantees to find the most localized allocation, *i.e.,* in the lowest subtree.

Bandwidth requirement of a valid allocation. Before presenting the allocation algorithm, we first ignore the time dimension and explain on how a fixed access bandwidth B per VM in a TIVC request translates into the bandwidth requirement on the internal links in the physical network.

If the N VMs required by a request can be found at a level-0 subtree, *i.e.,* within a physical machine, they can be allocated right away, as there should be enough bandwidth between the VMs in the same machine. Otherwise, the N VMs will reside in multiple subtrees, and the traffic between them will travel up and down the tree. This poses a subtle challenge as to how much bandwidth needs to be reserved on the tree links. Consider a link L that connects a left subtree containing m allocated VMs and a right subtree with $(N-m)$ VMs. Since each VM cannot send or receive at a rate more than B, the maximum bandwidth needed on link L is $min(m, N-m)*B$. Thus a valid allocation needs to satisfy $min(m, N-m)*B < R_L$, where R_L is the residual bandwidth of link L.

Now, taking the time dimension into account, a valid allocation for a TIVC request is an embedding of the VMs into a subtree of the datacenter where each link L connecting parts of the subtree satisfies the bandwidth requirement of each pulse and valley in the TIVC request. Specifically, for each $P_i = (T_{i1}, T_{i2}, B_i)$ in the TIVC request, link L should have enough residual bandwidth during interval (T_{i1}, T_{i2}), *i.e.,*

$$min(m, N-m) * B_i < R_L(T_{i1}, T_{i2}) \quad (1)$$

Efficient search via dynamic programming. As discussed above, when the VMs of a TIVC request cannot be satisfied within a rack, they may paternally be divided into subgroups which are then allocated out of different racks under a depth-1 subtree, and if not, out of different depth-2 subtrees, and so on. The cost for searching for such possible valid allocations can quickly become combinatorial. We make a key observation that a valid suballocation of K_1 VMs in a depth-$(d-1)$ subtree can be reused in searching for a valid suballocation of K_2 VMs, $K_2 > K_1$, in the parent depth-d subtree, and hence we can formulate the searching algorithm as a dynamic programming problem which runs very efficiently.

Finding a good allocation. Given a TIVC request, there can be many possible valid allocations in the physical network. There are two dimensions in the physical network that quantify a good allocation. First, in the vertical dimension, a good allocation should exhibit good locality, *i.e.,* the VMs allocated to it should be as

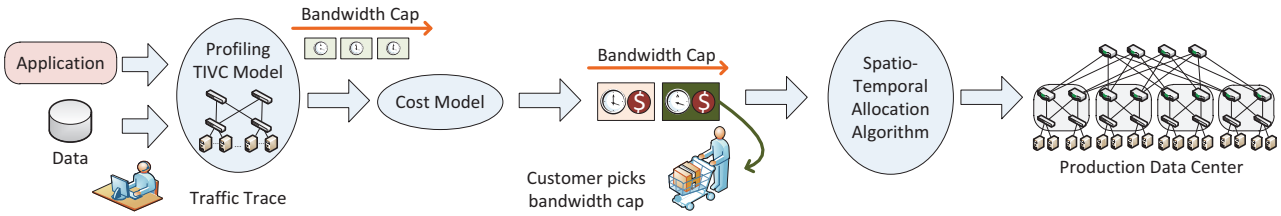


Figure 9: The Proteus system.

localized to a subtree as possible, as good locality conserves the bandwidth of the links in the upper levels of the tree. Second, it is possible that multiple equally localized allocations exist, *i.e.,* each within a depth-i subtree. A major consideration in choosing an allocation out of them is fragmentation, as allocating a TIVC to a subtree may result in the subtree having few VMs and little link bandwidth left to fit any future TIVCs. This fragmentation problem resembles the classic dynamic memory allocation problem in operating systems, for which a number of classic heuristic allocation strategies, including first-fit, best-fit, and worst-fit, exist. Our empirical experiments have shown there is no clear winner, and hence PROTEUS uses the local, first-fit strategy; it picks the first fitting lowest-level subtree out of all such lowest-level subtrees.

The algorithm. Figure 10 shows the TIVC allocation algorithm in pseudo code. The set of all possible numbers of VMs out of the N VMs needed by a job that can be allocated in the subtree rooted at v form the M set for v, M_v. Because of the bandwidth constraint (Equation 1), the numbers in M_v may not be continuous. Hence, we need to record the valid allocation out of each subtree during dynamic programming search, using a few data structures. Let $L_v[k]$ denote the set that contains the numbers of VMs that could be accommodated in the first k children of the subtree rooted at v, without considering the uplink bandwidth constraint of v. Then M_v contains all the values in $L_v[n]$ that can satisfy the uplink bandwidth constraint. To record the allocation in the traversed subtrees, *i.e.,* each possible value h in $L_v[k]$, we record in $D_v[k,h]$ the number of VMs assigned to the kth child of v, when it assigns h VMs in the first k children. The dynamic programming step is shown in lines 5–10, which calculates $L_v[k]$ and $D_v[k,h]$ recursively. Afterwards, all the candidate numbers of VMs are added to M_v if they pass the uplink bandwidth requirement of v (lines 11-14). If N can be allocated out of M_v, $Alloc()$ is called which performs recursion according to $D_v[k,h]$ while recording the bandwidth reservation of the relevant links, and eventually outputs the number of VMs per machine (level 0), and the algorithm terminates. We can easily show the algorithm outputs the first allocation (from left to right) that fits in the lowest-level subtree.

```
Algorithm: Allocation for TIVC request r
Input: Datacenter topology tree T
1. For level l from 0 to height(T)
2.     For each subtree v at level l
3.         If (l==0) L_v[0] = {0, ..., # avail. VMs} // leaf machine
4.         Else L_v[0] = {0}
5.         For v's child k from 1 to n
6.             L_v[k] = {0}
7.             For each possible value e in v's kth child's M set (M_{v_k})
8.                 For each possible value h in L_v[k − 1]
9.                     L_v[k] = L_v[k] ∪ {e + h}
10.                    D_v[k, e + h] = e
11.        M_v = ∅
12.        For each value h in L_v[n]
13.            If (bandwidth check of v's uplink per Eq. (1) == true)
14.                M_v = M_v ∪ {h}
15.        If N ∈ M_v
16.            Alloc(r, v, N)
17.            Return true
18. Return false
function Alloc(r, v, m):
19. If v is a machine
20.    allocate m VMs in v
21. Else
22.        For v's child k from n to 1
23.            Alloc(r, v_k, D_v[k, m])
24.            record bw reservation on v's kth link
25.            m = m − D_v[k, m]
```

Figure 10: The TIVC Allocation algorithm.

5.3 Enforcing TIVC Reservations

After allocating the VMs for a job, PROTEUS needs to configure the network elements to enforce the reserved bandwidth on the access links and internal links that connect the VMs. Prior works [11, 34, 26] have opted for an end-host only approach, which reserves per-VM access bandwidth in the hypervisor. To enforce reservations in in-network links, such approaches add significant complexity in the hypervisor, which needs to perform online rate monitoring for all VM pairs, communicate the rates to a centralized optimizer which calculates the max-min fairness for each VM pair, *etc.*, making such approaches less scalable. Bloating the hypervisors is also typically disliked since it adds complexity in the critical path and compromise robustness and security [13].

We make a key observation that even in a large-scale datacenter, the number of jobs that share a link at the same time is generally low. Consider a typical rack in the production environment with 40 machines and hence 160 VMs assuming conservatively 4 VMs per machine. If most jobs are small enough to fit within a rack, few jobs need to straddle the rack boundaries. On the other hand, very few large jobs which need to cross the core of the network can be scheduled to run concurrently since each of them consumes a large number of VMs. In our simulation runs over a datacenter with 16,000 machines (§6), we found fewer than 26 concurrent jobs per link (see Figure 17). Rate limiting such a low number of jobs sharing a link can be easily implemented using network switches available today. For example, the Cisco Nexus 7000 Series 32-port 10Gb module already supports up to 16K policers (for rate limiting 16K aggregates) [4] and 64K ACL entries (for defining the traffic aggregates). Furthermore, the above low number of jobs sharing a link, as well as the fact the edges of bandwidth pulses of different jobs happen at different times, suggest that reconfiguring the policers can be done with low overhead.

Multi-path routing. The TIVC allocation algorithm in §5.2 assumes a simple tree topology where the traffic between a VM pair follows a single path up and down the tree. However, datacenters may have networks with richer connectivities such as multi-rooted trees (*e.g.,* [27]) and fat-trees [7, 19]. These networks typically use hash-based or randomized techniques such as ECMP and Valiant Load Balancing to spread traffic across multiple equal cost paths, and can use more involved techniques such as Hedera [8] and MPTCP [31] to ensure uniform traffic spread despite flow length

variations. Therefore, such topologies can be incorporated into the TIVC allocation algorithm by treating the multiple links from each physical machine or switch that are used in multiple equal cost paths (to the same destination) as a single aggregation link, and enforcing bandwidth reservation of an aggregation link boils down to enforcing equal reservation split among the multiple physical links in the aggregate. We leave a detailed experimental study of such physical networks as future work.

6. EVALUATION

We use both simulations of large scale datacenter networks and our implementation on a testbed running real MapReduce applications to show PROTEUS exhibits significant advantage over a fixed-bandwidth reservation scheme such as Oktopus.

6.1 Simulation Setup

To show the effectiveness of PROTEUS in large scale datacenter settings, we developed a simulator that models VM and network bandwidth reservations in a shared datacenter. The simulator simulates a datacenter of three-level tree topology. There are 16,000 machines at level 0, each with 4 VM slots. 40 machines form a rack and are linked with a Top-of-Rack (ToR) switch with 1 Gbps links. Every 20 ToR switches are connected to a level-2 aggregation switch, and 20 aggregation switches are connected to the core switch of the datacenter. The default oversubscription of the physical network is 4, *i.e.,* ToR switches are connected to aggregation switches with 10 Gbps links, and aggregation switches to the core switch with 50 Gbps links.

Alternate abstractions. We compare PROTEUS with Oktopus, the state-of-the-art network abstraction [11]. Oktopus supports two network abstractions: virtual clusters (VC) and virtual oversubscribed clusters (VOC). However, VOC places a significant burden on the cloud users who not only have to specify the constant bandwidth constraint B, but also explicit subclustering of VMs that exhibit local communication and their oversubscription factors. Further, we do not observe any such communication locality in the applications we have studied. Thus we leave comparison with VOC as future work. Ideally, we should also compare TIVC with a baseline model that schedules jobs solely based on the number of available VMs. However, it is difficult to model in simulations the execution time elongation when jobs compete for networking freely.

Workload. We simulate tenant jobs based on the network workload extracted from the MapReduce applications studied in §2.2: Sort, Hive Join, and Hive Aggregation. We do not include Word Count as it has insignificantly low bandwidth requirement and will not benefit from bandwidth reservations provided by network abstractions like VC and TIVC. We use the no-elongation threshold bandwidth cap (§4.1) as the bandwidth requirement B under VC, and as the capping bandwidth in application profiling and model generation under TIVC. Using the same bandwidth cap B this way ensures that the job running times stay the same under the two abstractions during production runs. The generated TIVC parameters are shown in Table 2. To simulate a datacenter with diverse job mixes, we vary the number of VMs needed by each job; in our experiments by default the number of VMs per job request is exponentially distributed around a mean of 49 (following [11]).

6.2 Simulation Results

We compare PROTEUS with Oktopus under two scenarios: (1) A large number of tenant jobs are pooled at the job queue waiting to be scheduled to run. This workload captures production datacenters that host time-insensitive jobs, *e.g.,* data processing jobs to be run overnight. (2) Tenant jobs arrive dynamically and are accepted only if they could be scheduled at the moment of arrival. This workload is representative of shared clouds that host time-sensitive jobs. For each scenario, we first simulate 5,000 jobs running a single application, and then simulate 5,000 jobs of equally mixed applications. The four workloads are denoted as Sort, Hive Join, Hive Aggregation and Mixed.

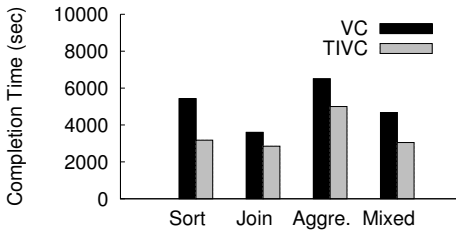


Figure 11: Total completion time.

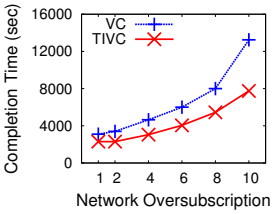


Figure 12: Completion time with varying oversub. for mixed workload.

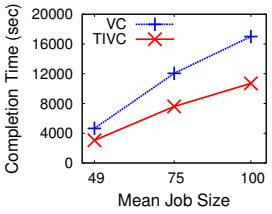


Figure 13: Completion time with varying mean job size for mixed workload.

6.2.1 *Batched Jobs*

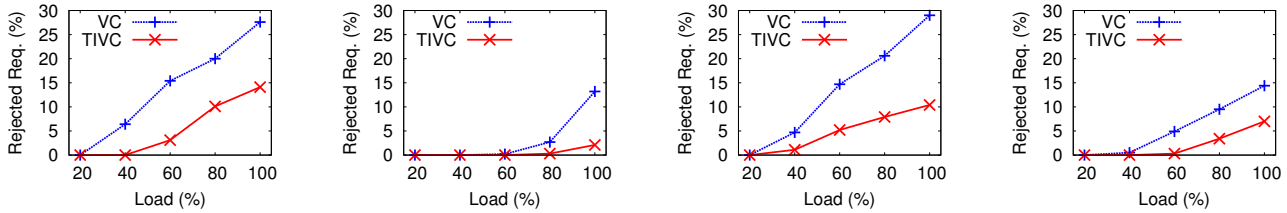
For batched jobs, the job scheduling policy tries to maximize the job throughput. Under VC, this is achieved by going through the job queue and schedule the jobs that can be scheduled to run, whenever a job is finished. Under TIVC, however, this can be inefficient, as before any running job is finished, there can be enough residual bandwidth freed up so that a new job can be scheduled. Instead, the job scheduler rescans the job queue every 10 seconds, which is 20% of the average inter-job completion time.

Job completion time. Figure 11 plots the time to complete all 5,000 jobs under VC and TIVC. We see for all workloads, TIVC significantly improves the completion time, and hence job throughput of the datacenter, over VC. In particular, compared to VC, TIVC reduces the completion time by 41.5%, 20.8%, 23.1%, and 34.5% for Sort, Hive Join, Hive Aggre., and Mixed, respectively.

Varying the oversubscription rate and job size. We repeat the above experiments with varying oversubscription rates in the physical datacenter network. Figure 12 shows TIVC provides greater advantage over TC when the oversubscription rate is larger, reducing the total completion time by 35.2%, 36.0%, and 41.5% under oversubscription rates 6, 8, and 10, respectively. Similarly, Figure 13 shows increasing the mean job size N further increases the performance advantage of TIVC over VC since larger jobs are more likely to traverse the oversubscribed core network links.

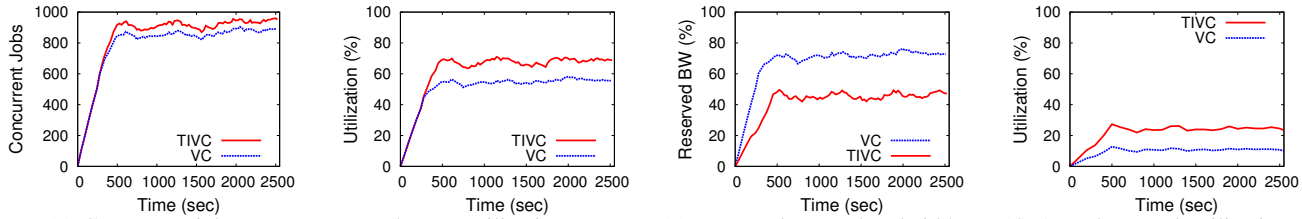
6.2.2 *Dynamically Arriving Jobs*

We now consider the cloud scenario where job requests arrive over time. Assume the job arrival follows a Poisson process with rate λ, then the load on a datacenter with M VMs total is $\lambda \cdot N \cdot Tc/M$ where N is the mean job request size (*i.e.,* 49) and Tc is the mean job completion time. If a job cannot be allocated upon its arrival, it is rejected, as is the case with Amazon's EC2 job admission control [1] today. We again simulate 5,000 job requests under VC and TIVC while varying the load factor.



(a) Sort. (b) Hive Join. (c) Hive Aggregation. (d) Mixed.

Figure 14: Percentage of rejected requests with varying datacenter load.



(a) Concurrent jobs. (b) VM utilization. (c) Reserved access bandwidth. (d) Actual network utilization.

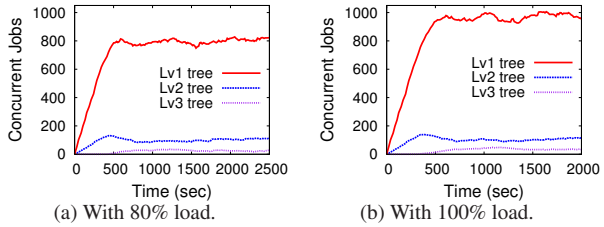
Figure 15: Concurrent jobs, VM utilization, reserved and actual utilization of bandwidth, for Mixed and 80% load.

Job rejection rate. Figures 14(a)-14(d) plot the rejection rates for the three application workloads and the mixed workload. We observe that under low load, *e.g.,* 20%, the total networking reservation under both VC and TIVC can be met and hence both accept all jobs. As the load increases, VC rejects far more requests than TIVC. For example, at 80% load, 20.0%, 2.7%, 20.6%, and 9.5% of the requests are rejected under VC compared to 10.1%, 0.3%, 7.9%, and 3.4% under TIVC, for the four workloads, respectively.

Job concurrency and VM/network utilization. To understand how TIVC achieves much lower rejection rates than VC, we look at the number of concurrent jobs scheduled and the VM and network utilization under the two models. Due to page limit, we only show the results for the mixed workload under one load factor, 80%. Figure 15(a) shows that after the initial job arrival ramp-up phase, TIVC consistently achieves about 7% higher job concurrency than VC. Since the extra jobs accepted by TIVC tend to be larger than average, the 7% higher job concurrency under TIVC translates into on average close to 13% higher VM utilization (of the total 64,000 VMs in the datacenter) than under VC, as shown in Figure 15(b).

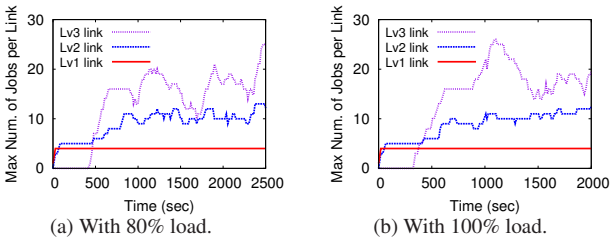
Finally, the reason TIVC is able to fit more jobs is by exploiting lower networking periods of a job to schedule other jobs. Figure 15(c) shows the average reserved access bandwidth over time under VC and TIVC. We see VC reserves on average 26.4% (of the link capacity) higher bandwidth than TIVC. However, Figure 15(d) shows TIVC achieves on average about 20.1% actual network utilization, calculated by adding the instantaneous traffic demand of individual jobs over each access link, and then averaged over all access links, much higher than the 8.9% under VC. This confirms that by capturing the time-varying nature of application traffic demand, TIVC is able to achieve much more efficient bandwidth reservation than VC. We note the overall low actual network utilization under the explicit network reservations may seem counter-intuitive. This is precisely the price to pay for predictable performance, *i.e.,* to reserve enough bandwidth so that the execution of real world applications, which can have diverse, bursty traffic phases, is not elongated (§4.1).

Job locality and link sharing. To assess the spatial locality of the TIVC jobs allocated by PROTEUS, we plot the average number of concurrent jobs allocated at different subtree levels in the datacenter, for the mixed workload runs. Figure 16 shows that after the ramp-up phase, under 80% load, on average around 795 out of the 920 total concurrent jobs are allocated within level-1 subtrees,



(a) With 80% load. (b) With 100% load.

Figure 16: Number of concurrent jobs allocated to diff. tree levels for Mixed.

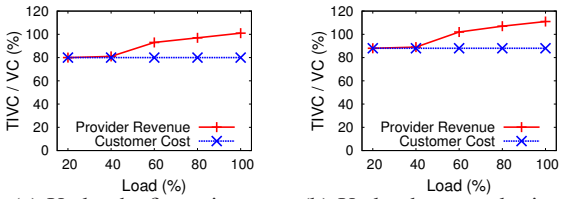


(a) With 80% load. (b) With 100% load.

Figure 17: Max number of jobs sharing a link at diff. tree levels for Mixed.

i.e., confined to ToR switches, and on average fewer than 99 and 26 jobs are within level-2 and level-3 subtrees, respectively. Under 100% load, there are fewer than 102 and 37 jobs within level-2 and level-3 trees. These numbers confirm that the vast majority of the jobs are localized to small subtrees. The immediate consequence of such locality is that there are few jobs sharing any link at the same time in the datacenter network. Figure 17 shows the maximum numbers of jobs sharing a link are less than 13 and 26 at level 2 and 3, and stay at 4 at level 1 (*i.e.,* the 4 VMs of a machine are allocated to 4 different jobs), under 80% and 100% load. We also measured the locality in separate experiments when the workload has only large jobs, *i.e.,* requiring thousands of VMs, and when the workload has mixed small and large jobs, and again found there are few jobs sharing any given link in the whole network. The reason is when jobs are large, few can be scheduled to run concurrently. In summary, the low number of jobs sharing any link in the network despite job size mixes suggests per-job bandwidth reservation in the internal links can be easily accomplished using policers in off-the-shelf switches (§5.3).

Tenant cost and provider revenue. Today's cloud providers such as Amazon EC2 charge tenants solely based on the consumed VM-



(a) Under the first price. (b) Under the second price.

Figure 18: Relative provider revenues and tenant costs for mixed workload.

time. In this case, the fraction of job requests that are accepted and the costs for them determine the cloud provider's revenue. Since the number of VMs allocated to each job and its execution time stay the same under VC and TIVC, the lower rejection rate under TIVC compared to VC directly translates into increased revenue for today's cloud provider, while the individual tenant's cost stays the same. For example, for the mixed workload in Figure 14(d), VC rejects 4.6%, 6.1%, and 7.4% more jobs under 60%, 80%, and 100% load, respectively. Since the rejected jobs tend to be larger than average, the extra rejected jobs translate into 16%, 22%, and 27% lower provider revenue under VC.

We envision that tomorrow's cloud providers will and should explicitly charge for networking bandwidth in providing tenants with explicit bandwidth reservations such as VC and TIVC. Since developing fair yet efficient charging model is still ongoing research [30, 10], we adopt the simple charging model in [11] which effectively charges networking based on the total reserved bandwidth volume over time in such a way that the cloud provider (*e.g.,* Amazon EC2) remains revenue neutral in transitioning from the VM-only charging model to the new model. Specifically, a tenant using N VMs for time T will be charged $N(T \cdot k_v + k_b \cdot V)$, where k_v is the unit-time VM cost, k_b is the unit-volume bandwidth cost, and V is the total bandwidth volume reserved over the time period T.

Under the above charging model, we compare the cloud provider revenue and the tenant cost under VC and TIVC for two sample estimated k_v and k_b prices: (0.04$/hr, 0.00016$/GB) and (0.04$/hr, 0.00008$/GB), for the mixed workload run in Figure 14(d). We calculate the ratio of the total cloud provider revenue and the ratio of the tenant job cost, under TIVC versus under VC. Figure 18(a) shows the two ratios under the first price. We see that TIVC allows tenants to pay on average about 20% less than VC, for accepted jobs, independent of the load, from reduced network usage. At low load, *e.g.,* 20–40%, the cloud provider revenue under TIVC is about 20% lower than under VC because the cloud provider accepts almost all jobs under both schemes while the tenants under TIVC on average pay 20% less than under VC. At close to 100% load, however, not only do tenants under TIVC pay 20% less than under VC, the cloud provider stays revenue neutral under TIVC compared to under VC. This is because the provider is able to accept about 7% more jobs under TIVC than under VC, which corresponds to about 13% higher VM utilization (again since these extra jobs tend to be larger than average). Finally, Figure 18(b) shows under the second price, TIVC not only allows tenants to pay on average about 12% less than under VC at all loads, but also allows the cloud provider to make more revenue when the load crosses 60%.

6.3 Testbed Experiment

We implemented PROTEUS following the description in §5 on a datacenter testbed consisting of 18 machines (specification in §2.2) forming a 3-tier tree topology as shown in Figure 19. Each machine runs 2 VMs, and the testbed switches are implemented using servers with NetFPGA cards with 4 1Gbps ports. We use the rate limiter module provided in the base package of NetFPGA reference router to limit the capacity of each internal tree link to emulate an oversubscribed datacenter. The level-1 (*i.e.,* between the machines and ToR switches), level-2, and level-3 link capacities are 230, 700, and 1000 Mbps, respectively.

The bandwidth provisioning for access links is implemented via the Linux traffic control API `tc`. For internal links, since it is not straight-forward to implement per-job rate limiters, we configured the combined rate of jobs allocated to the sub-tree. For example, assuming nodes $1-6$ are allocated to jobs 1 and 2, then the link between A and G, shared between the two jobs, would be configured with the sum of bandwidth requirements of the two jobs.

We use a mixed workload of 30 jobs, 10 each of the three applications, Sort, Hive Join, and Hive Aggre. PROTEUS profiles the 3 applications on the testbed, generates the TIVC and VC models using no-elongation threshold bandwidth cap, and allocates the jobs accordingly. We also run the same 30 jobs under a baseline model, which schedules the jobs solely based on the number of available VMs. Figure 20 shows the completion time for the workload are 2405, 3770, 5140 seconds, for Baseline, TIVC, and VC, respectively. TIVC reduces VC's completion time by 27% from more efficient bandwidth reservation and hence scheduling. However, Baseline has the shortest completion time since it aggressively schedules jobs to compete freely for the network which results in higher overall networking utilization, but however can lead to unpredictable application performance. Figure 21 shows the CDF of the per-job execution time under Baseline and under VC relative to that under TIVC. We see that in the median case, per-job execution time under Baseline is 10% longer than that under TIVC. Except for a few variations, TIVC results in similar per-job execution times as VC, because both models reserve the threshold bandwidth and avoid unpredictable competition for the network.

To evaluate the scalability of the TIVC allocation algorithm, we measure the time to allocate each of the 5,000 job requests in the large datacenter with 64,000 VMs used in §6.1. Our allocation algorithm is highly scalable; the single-threaded code running on an 8-core Intel Xeon E5410 2.33 Ghz processor and 16 GB RAM has a median time of 18.0ms and the 99^{th} percentile time of 28.0ms.

7. RELATED WORK

Our work is closely related to the recently proposed virtual network abstractions [20, 11, 33]. We discussed [20, 11] in detail in §2. Like Oktopus, Gatekeeper [33] also proposes a per-VM hose model but for full bisection networks and focuses on managing servers' access bandwidth. The hose model was originally introduced in [18] for wide-area VPNs and did not consider allocating physical or virtual machines. Compared to these work, our work proposes TIVC, which extends the per-VM hose model to model the time-varying nature of networking requirement of cloud applications. More importantly, our work takes the first step towards automatically deriving the model parameters for a representative class of cloud applications.

Our work is also related to previous work on mechanisms for sharing datacenter networks. As discussed in §5.3, most of previous work (*e.g.,* [11, 33] use a hypervisor-based framework for enforcing bandwidth reservation in the network which can suffer poor scalability. Seawall [34] and Netshare [26] propose bandwidth slicing mechanisms that aim to provide fair sharing of networks with minimum bandwidth guarantee and statistical multiplexing, but do not provide deterministic bandwidth guarantees.

PROTEUS shares the same profiling methodology with Elastisizer [22], StarFish [23] and CBO [21] which focus on choosing the type and number of VMs for MapReduce jobs to balance cost/performance objectives. These work ignore networking requirement and hence complement our network profiling technique.

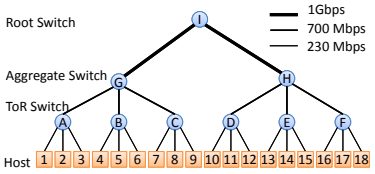


Figure 19: Testbed topology.

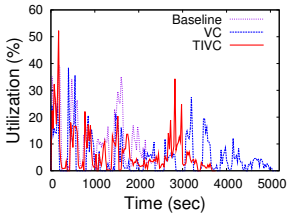


Figure 20: Network utilization.

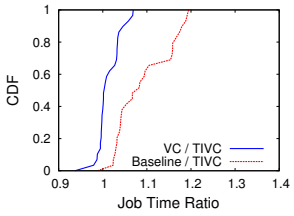


Figure 21: CDF of relative job time.

Our TIVC allocation algorithm is related to previous work on virtual network embedding (*e.g.,* [15, 35]) and testbed mapping [32]. These work resort to heavy-weight optimization solvers such as linear programming and can not scale to the larger number of VMs in modern datacenters.

Finally, several very recent work [30, 10] start to study fair and efficient charging model in sharing datacenter networks. These studies are complementary to PROTEUS and can be incorporated into PROTEUS in choosing cost-effective TIVC models.

8. CONCLUSIONS

In summary, the primary contributions of this paper are the design of the first network abstraction (to our best knowledge), TIVC, that captures the time-varying nature of cloud applications, and a systematic profiling-based methodology for making the abstraction practical and readily usable in today's datacenter networks. Our experimental evaluation using real MapReduce applications shows that TIVC significantly outperforms previous fixed-bandwidth network abstractions in improving job throughput and hence cloud provider revenue and reducing tenant cost. Our work takes a significant step forward towards efficient and cost-effective sharing of datacenter networks in providing cloud customers with predictable performance and cost.

The PROTEUS system which implements the TIVC abstraction can be readily used to extend today's dominant utility computing model offered by public clouds, which requires the customers to explicitly request for, and manage, virtual machines for their jobs, to support an *extended utility computing model* that directly meets the service time objectives of cloud customers. In this model, PROTEUS directly allocates an application slice of the datacenter, *i.e.,* a TIVC specification, that meets the target service time of a given application at the minimum cost to the customer.

Acknowledgements

We thank the anonymous reviewers, Chuangxiong Guo, and especially our shepherd, Ant Rowstron, for their helpful comments. We thank Pawan Prakash for his help with the testbed experiments. This work was supported in part by NSF grants CNS-1054788 and CRI-0751153.

9. REFERENCES

[1] Amazon ec2 api ec2-run-instances. http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/ApiReference-cmd-RunInstances.html.
[2] Apache hive. http://hive.apache.org/.
[3] Apache pig. http://pig.apache.org/.
[4] Cisco nexus 7000. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9402/ps9512/Data_Sheet_C78-437757.pdf.
[5] Hive performance benchmarks. https://issues.apache.org/jira/browse/HIVE-396.
[6] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou. Re-Optimizing data-parallel computing. In *Proc. of USENIX NSDI*, 2012.
[7] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. of ACM SIGCOMM*, 2008.
[8] M. Al-fares, S. Radhakrishnan, et al. Hedera: Dynamic flow scheduling for data center networks. In *Proc. of USENIX NSDI*, 2010.
[9] 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.
[10] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. The price is right: Towards location-independent costs in datacenters. In *Proc. of ACM HotNets*, 2011.
[11] H. Ballani, P. Costa, T. Karagiannis, and A. I. T. Rowstron. Towards predictable datacenter networks. In *Proc. of ACM SIGCOMM*, 2011.
[12] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursiv query rocessing strategies. In *In Proc. of SIGMOD*, 1986.
[13] C. Brenton. Hypervisor vs host based security. https://cloudsecurityalliance.org/wp-content/uploads/2011/11/hypervisor-vs-hostbased-security.pdf.
[14] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. HaLoop: Efficient Iterative Data Proecssing on Large Clusters. *Proc. of the VLDB Endowment*, 3(1), 2010.
[15] N. Chowdhury et al. Virtual Network Embedding with Coordinated Node and Link Mapping. In *IEEE INFOCOM*, 2009.
[16] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears. Online aggregation and continuous query support in mapreduce. In *ACM SIGMOD*, 2010.
[17] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of USENIX OSDI*, December 2004.
[18] N. G. Duffield, P. Goyal, A. Greenberg, et al. A flexible model for resource management in virtual private networks. In *Proc. of ACM SIGCOMM*, 1999.
[19] A. Greenberg, J. R. Hamilton, et al. VL2: a scalable and flexible data center network. In *Proc. of ACM SIGCOMM*, 2009.
[20] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, et al. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proc. of ACM CoNEXT*, 2010.
[21] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *PVLDB*, 4(11):1111–1122, 2011.
[22] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proc. of ACM SOCC*, Oct. 2011.
[23] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *Proc. of CIDR*, 2011.
[24] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: a platform for fine-grained resource sharing in the data center. In *Proc. of USENIX NSDI*, 2011.
[25] J. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999.
[26] T. Lam and G. Varghese. Netshare: Virtualizing bandwidth within the cloud. *UCSD Technical Report*, 2009.
[27] J. Mudigonda, P. Yalagandula, M. Al-Fares, and H. L. Jeffrey Mogul. SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies. In *Proc. of USENIX NSDI*, 2010.
[28] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand. Ciel: a universal execution engine for distributed data-flow computing. In *Proc. of USENIX NSDI*, 2011.
[29] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. *Technical Report 1999-66, Stanford Infolab*, 1999.
[30] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: Sharing the network in cloud computing. In *ACM HotNets*, 2011.
[31] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *Proc. of ACM SIGCOMM*, 2011.
[32] R. Ricci, C. Alfeld, and J. Lepreau. A Solver for the Network Testbed Mapping Problem. *SIGCOMM CCR*, 33(2), 2003.
[33] H. Rodrigues et al. Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. *HP Technical Report*, 2011.
[34] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proc. of USENIX NSDI*, 2011.
[35] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2), 2008.
[36] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*, 2010.