

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	$\langle N, 382s, 4Mbps, (17s, 202s, 400Mbps) \rangle$	22.8%
Hive Join	4	$\langle N, 672s, 50Mbps, (46s, 183s, 100Mbps), (284s, 301s, 300Mbps), (329s, 363s, 300Mbps), (383s, 413s, 300Mbps), (434s, 464s, 300Mbps), (485s, 517s, 300Mbps), (539s, 634s, 100Mbps) \rangle$	17.6%
Hive Aggre.	4	$\langle N, 535s, 4Mbps, (27s, 253s, 60Mbps), (268s, 492s, 350Mbps) \rangle$	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 pro-

gramming 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

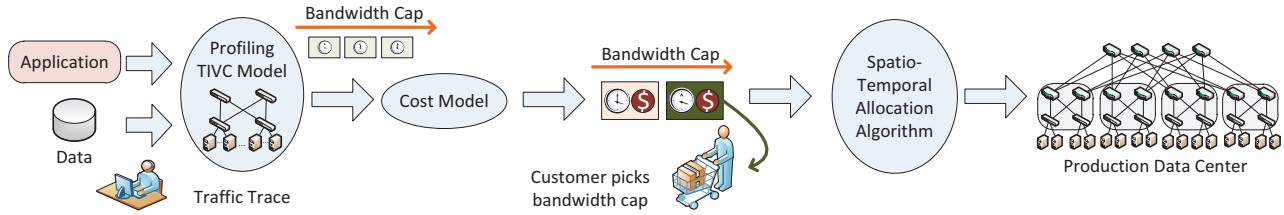


Figure 9: The Proteus system.

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  $k$ th 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.

### 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

---

**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** ( $l==0$ )  $L_v[0] = \{0, \dots, \# \text{ avail. VMs}\}$  // 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  $k$ th child's  $M$  set ( $M_{v_k}$ )
  8.         **For** each possible value  $h$  in  $L_v[k-1]$
  9.          $L_v[k] = L_v[k] \cup \{e+h\}$
  10.          $D_v[k, e+h] = e$
  11.        $M_v = \emptyset$
  12.       **For** each value  $h$  in  $L_v[n]$
  13.         **If** (bandwidth check of  $v$ 's uplink per Eq. (1) == true)
  14.          $M_v = M_v \cup \{h\}$
  15.       **If**  $N \in M_v$
  16.          $Alloc(r, v, N)$
  17.         **Return** true
  18.       **Return** false
  19. **function**  $Alloc(r, v, m)$ :
  20.   **If**  $v$  is a machine
  21.     allocate  $m$  VMs in  $v$
  22.   **Else**
  23.     **For**  $v$ 's child  $k$  from 1 to  $n$
  24.        $Alloc(r, v_k, D_v[k, m])$
  25.       record bw reservation on  $v$ 's  $k$ th link
  26.        $m = m - D_v[k, m]$
- 

Figure 10: The TIVC Allocation algorithm.

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

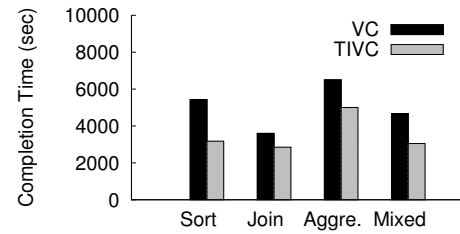


Figure 11: Total completion time.

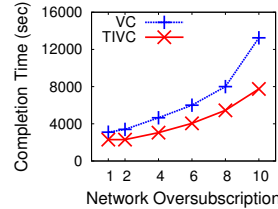


Figure 12: Completion time with varying oversub. for mixed workload.

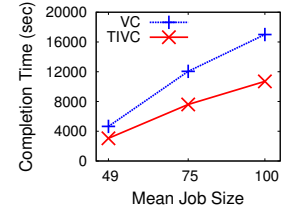


Figure 13: Completion time with varying mean job size for mixed workload.

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.

#### 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 VC 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  $\lambda$ , then the load on a datacenter with  $M$  VMs total is  $\lambda \cdot N \cdot T_c / M$  where  $N$  is the mean job request size (*i.e.*, 49) and  $T_c$  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.

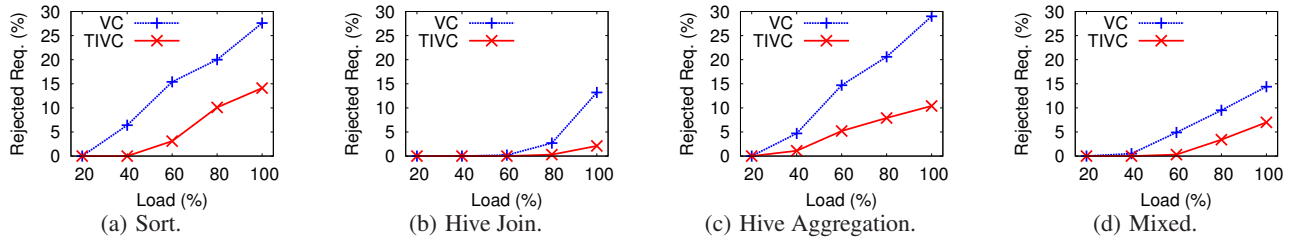


Figure 14: Percentage of rejected requests with varying datacenter load.

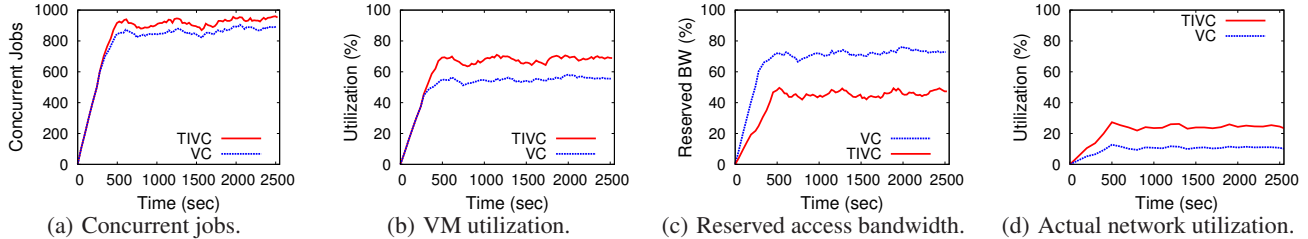


Figure 15: Concurrent jobs, VM utilization, reserved and actual utilization of bandwidth, for Mixed and 80% load.

**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,

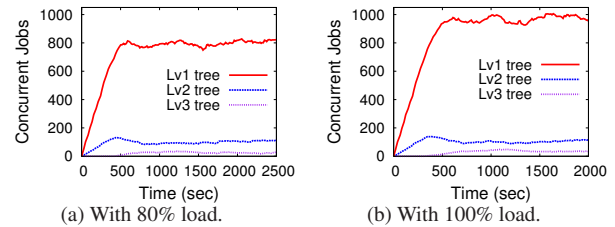


Figure 16: Number of concurrent jobs allocated to diff. tree levels for Mixed.

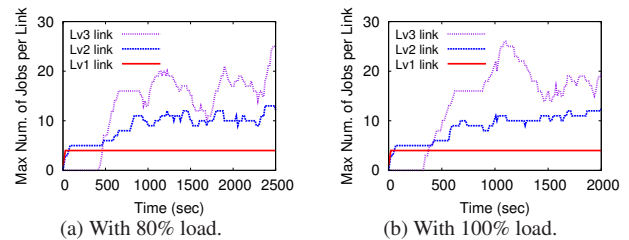
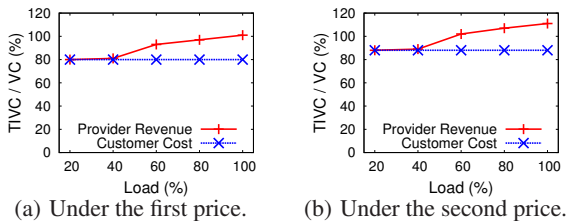


Figure 17: Max number of jobs sharing a link at diff. tree levels for Mixed.

*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-



**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 Gbps 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<sup>th</sup> 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.

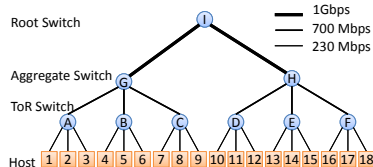


Figure 19: Testbed topology.

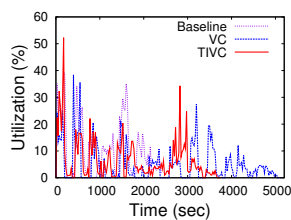


Figure 20: Network utilization.

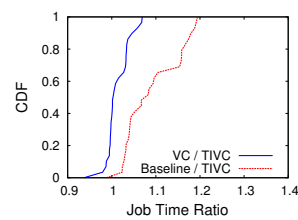


Figure 21: CDF of relative job time.

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](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 recursive query processing 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 Processing 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.