INTERNATIONAL COMPUTER SCIENCE INSTITUTE

STEM+: Allocating Bandwidth Fairly To Tasks

Andrey Lukyanenko [§*ǂ], Ilya Nikolaevskiy [§*ǂ], Dmitriy Kuptsov [§*ǂ], Andrei Gurtov [§*ǂ], Ali Ghodsi [ǂ], and Scott Shenker [*ǂ]

TR-14-001

April 2014

Abstract

Fair sharing of bandwidth among tenants in datacenters is important to guarantee prompt execution while providing isolation between different jobs. Existing bandwidth allocation methods lack a concept of a task reflecting the dependency between allocations on links. Moreover, existing approaches do not consider the tenants to be smart individuals and lack understanding of a threat that strategic players can produce. In this work we introduce a Strategy-proof Task-Enforcement Mechanism (STEM) which is the only strategy-proof mechanism for datacenter allocation. It seamlessly utilizes task-aware models. While tenants are able to improve their allocations by relocating demands among links, it also improves the global allocation resulting into a strong Nash equilibrium among tenants. This is in contrast to pricing or Competitive Equilibrium from Equal Incomes (CEEI) which permits tenants to inflate their demands and in some cases loosing sharing-incentives. We extend STEM with STEM+ -a work-conserving allocation mechanism.

§ Helsinki Institute for Information Technology HIIT, Finland
* International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, California, 94704
ǂ University of California, Berkeley

This research is supported by TEKES as a part of the Future Networks, Society and Modeling - FuNeSoMo program.

1. INTRODUCTION

Recent works on bandwidth allocations in datacenter suggest multiple solutions; many of them provide some of the desired properties of bandwidth guarantees, work-conservation and fairness. However, performance, as a flow completion time, seems to be of a higher value for production networks[1] rather than fairness. An anecdotal example of difference between performance (FIFO) and fairness (process sharing) appears in different works [20, 11, 15]. Fairness is not a proclaimed goal in production networks, rather fairness plays the role of a mechanism weakening requirement for global job scheduling.

Non-production networks[2], as opposed to production networks, define performance differently. There is no single administrative domain anymore, no strict behavioral control or same utility function. Fairness in cloud plays an additional behavior-enforcement role. Cloud, while aiming at network isolation, benefits from sharing incentives, i.e. if a tenant of a cloud does not require as much bandwidth at one path as she requires bandwidth at another, then she can willingly reduce the flow rate at the first path in exchange for additional available rate at the other path. Moreover, the exchange of one capacity for another can be not equivalent, but still beneficial. Thus, in cloud fairness plays the main role for incentivizing tenants to behave in the public network. Many studies on production networks as well as public cloud networks don't address the strategic properties of the tenants at all. Some works address the simplest strategies, e.g., rate control misbehavior. We observe that because of that many solutions lack strategy-proofness i.e., misbehaving tenants can gain more than behaving.

Task-aware allocation is yet another topic that gets less attention than it should. While generally it is understood that "stragglers" impact and even define network applications' performance, such as MapReduce, to our best knowledge, there is no work on the study of tenants utility based on task-aware models. The task-aware models studied are rather limited by their hose-models representations [3, 18, 21], i.e., models requiring bandwidth guarantees expressed in forms of hose-models; after the allocation each flow utilizing a part of the hose-model-based share in its own fashion.

While it is tempting to use local individual flow allocations, as it requires small to no coordinations, a task-aware allocation model needs to provide better understanding of the network. One of the drawbacks of task-aware model is requirement of coordination, but later we will show that otherwise misbehavior is always possible. CoFlow is a recent work that introduces a form of task-aware model [9], the presented model is fully valid in our case but slightly differs from ours (§3). CoFlow also defines API which exposes task semantic to the network. In this work we introduce an algorithm that does not require such API, but can in an obvious way benefit from it.

To address the described problems and challenges we want to introduce an allocation protocol, which has the following properties:

1. **Seamless task-awareness.** The allocation needs to address problems for task-aware application, but it should work for individual flows as well.

2. **Strategy-proofness.** The tenants should be treated as strategic individuals, which are able to misbehave. A strategy-proof mechanism de-incentivize, the misbehavior.

3. **Sharing incentives.** Sharing is more beneficial than non-sharing, i.e., the whole capacity is divided equally for all tenants and tenants are fully isolated.

4. **Work conservation.** This is a form of high utilization guarantee for the network. As in a task-aware allocation not the whole network usage is required, work conservation can distribute unused resources at a run-time.

5. **Fairness.** The tenants should have understanding of fairness of their allocations, if they trade some of own capacity they should understand what they get for the lost capacity.

6. **Backward compatibility.** Avoid any changes to the infrastructure (hardware).

The first obvious solution that satisfies at least a few of the above properties is to use any pricing mechanism as suggested in [19]. However, we show that pricing breaks the sharing incentive property in some cases, and leads to misbehavior in general (§4). Also we show that no local (task-unaware) algorithm can satisfy strategy-proofness property.

Finally, to achieve the desired properties we introduce the Strategy-proof Task-Enforcement Mechanism (STEM). We use dominant resource fairness (DRF) [13] as fairness metric for STEM, however DRF is not only option available. Moreover, we find that strategy-proofness cannot be achieved with work conservation, thus we introduce a work-conserving extension for the architec-

[1]Instead of "Production network" we can name such networks as private data center network running single company's jobs or having extensive administrative control over tenants. However, we prefer to avoid such bulky naming especially that many production networks are exactly like that.

[2]Here and further we use "non-production network" or "[public] cloud" meaning an infrastructure-as-a-service (IaaS) architecture (many of discussions apply to PaaS, SaaS, etc.), where tenants are not controlled by the same administrative domain and have some control over the network.

ture with STEM+ at the cost of weakening the strategy-proofness property.

The rest of the paper is organized as follows. Section 2 provides the motivation for our work. In Section 3 we introduce the task model for bandwidth allocation and define our model. Section 4 describes the proposed bandwidth allocation mechanism. In Section 5 we analyze a datacenter oriented model from a game-theoretic perspective and show the existence of equilibria for CEEI and DRF allocations. In Section 6 we present results of trace-driven simulation. In Section 7 we give insights on implementation of DRF bandwidth allocation in datacenter. In Section 8, we briefly review the related work. Section 9 concludes the paper.

2. TENANT STRATEGY SPACE AND MOTIVATION

The previous work on bandwidth allocation in datacenters primary is focused on performance [15, 21], minimum bandwidth guarantees [3, 21, 23] and local fairness [20] without any explicit study of behavior, strategy space and "happiness" of tenants. To show that the tenants have different network experience based on their sizes, we will use state-of-the-art algorithms from existing literature, namely proportional sharing at link-level (PS-L) and proportional sharing at network-level (PS-N) [20]. Both of them aim at achieving fairness using weighted fair queuing (WFQ) at switches, the difference is only on the weights which each mechanism assigns to tenants. For PS-L the weight is the number of VMs which flows[3] are going through current switch, while for PS-N the weight is simply the total number of VMs that the current tenant has in the cloud.

Our intuition is that there is no local fair allocation algorithm which is capable of achieving global fairness for different types of tenants; although PS-N seems to be aiming exactly at that, because it uses the tenant size for WFQ. To support our intuition we run a trace-based simulation with data from 3200-node Facebook production data center [10] (§6 has details about the setup). For that analysis we introduced three classes of differently sized tenants, where the size is defined as the total number of VMs of the tenant: (i) A for interval [0,150), (ii) B for [150,300), and (iii) C for [300,450). The flow allocation results are shown in Figure 1 as CDFs.

For both policies, PS-L and PS-N, we observe that the tenants form three separate clusters of performance based on their sizes. The flows of lowest class A gain the most resources, while the flows of largest class C gain the least. Moreover, the 80% of flows of class C perform not better than 10% worst-performing-flows of A. On average each flow of the class A is 10 times faster than class C. This observation is quite contradictory to the fact that the worst performing tenants are the highest contributors to the system, as each tenant commonly pays on per VM usage.

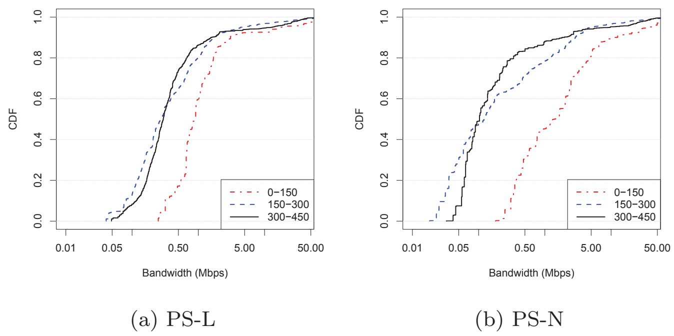


Figure 1: Tenant classes produced with local allocation policies

On another hand, large tenants have more control freedom in the network, they can influence the part of the network which small tenants cannot. For *all* tenants we observe the following strategic ways to improve their allocations:

1. **Manipulating flow data rates**, e.g., by violation of TCP congestion control rates. This is well understood misbehavior in the literature as well as solutions are known (rate control at hypervisor or observation of misbehaving flows).

2. **Splitting the task in space or time.** Some jobs can be split into smaller ones and then united, if the current allocation algorithm favors smaller job sizes. At the same time some on-line profiling algorithms favor small traffic volumes (by discriminating elephant flows). Thus tenants can benefit by splitting jobs in space or time into subjobs and after finalizing unite them again.

   E.g. let $f(k)$ be the function of utility for the job size k, then we introduce another utility function $\hat{f}(k)$ as $\hat{f}(K) = max[f(K), \max_I\{\hat{f}(K \setminus I) + \hat{f}(I) - g(I, K \setminus I)\}]$, where g is the cost for uniting subjobs I and $K \setminus I$. Condition $\hat{f}(N) > f(N)$, where N is the total size of the job, is true when splitting a job is more beneficial then executing it as one unit.

3. **Extra blocking traffic.** Consider Figure 2. There are three tenants (A,B,C) and two bottlenecked identical links (L_1,L_2). L_1 is shared by tenants B

[3]Here we consider that a unique pair of VM-to-VM can create only one flow. Technically, they create multiple parallel TCP flows, but from our perspective this will be one batch of flows, each individual TCP flow will get a fraction of the batch as throughput. In this paper by "flow" we mean this batch flow, if it is not said otherwise.

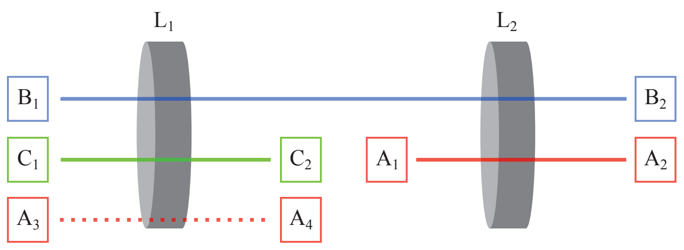


Figure 2: Tenant A can manipulate the allocation by creating extra flows. E.g., in order to reduce flow $B_1 \rightarrow B_2$, tenant A creates a flow $A_3 \rightarrow A_4$.

and C. It also can be used by A, although there is no need in that traffic for tenant A. L_2 is shared only by tenants B and A. L_1 is shared equally between B and C, L_2 is shared equally between A and B. Now let us assume that A creates jamming traffic at link L_1, then three tenants start to share it equally. As a consequence the rate at which tenant B is using the link will reduce to the same value ($\frac{1}{3}L_2$, as $L_1 = L_2$). This will result in acquiring $\frac{2}{3}L_2$ by tenant A. So it turned out that this misbehavior is beneficial for tenant A. **Extra blocking traffic is always possible with work conservation property.**

4. **Changing the roles of VMs.** This is rather simple, when a part of mappers, for example, becomes reducers and otherwise. It can allow a tenant to localize traffic, etc. To do so only a program (parameter) that is run at local machines needs to be changed, see Figure 3. There is one straggled flow $A_1 \rightarrow A_2$. Tenant A can get rid of the straggler simply by switching the roles of A_2 and A_3.

5. **Other solution-specific misbehavior/strategies.** For example, one of the recent works on a task-aware algorithm suggests to order flows by their creation time and based on that organize priority in queues [11]. The simplest way to misbehave in that case would be pre-creation of multiple flows which may be used in future (i.e., capture high priority values as fast as possible).

We observe that previous works were oblivious to the aforementioned strategic (mis-)behavior, which is quite a reasonable assumption for production networks. However, production networks rarely have fairness as a key criteria for performance. Public clouds have completely different situation. Some of the previous work [20] mentions the strategy-proofness problem in the cloud, but leaves an assumption that at application-level abstraction tenants cannot influence allocations, i.e., allocations are uncontrollable from Hadoop API. However, this is a completely invalid argument for IaaS, as any

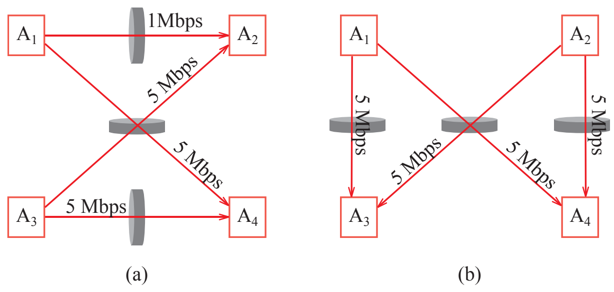


Figure 3: Tenant A can get rid of "straggling" flow $A_1 \rightarrow A_2$ by changing the roles of VMs. Traffic patterns (a) and (b) are isomorphic to each another.

new application mechanism coming to the market can easily break this assumption.

Big tenants observing the unfair flow distribution in Figure 1 above and having big strategic space have a strong incentive to start misbehaving, e.g., by using strategic options presented above.

3. MODELING

First of all, we formally study the properties of the allocation models in consideration. First we introduce the task model we employ – an essential connection between flow performance requirements, then we define the network and application graph model, finally we introduce the utility function which we will use to compare performance and fairness of allocations.

3.1 Task model

Our entire model revolves around the notion of tasks. We are not claiming the novelty for the task model at flow level here. We note that some motivation for our work can be found in CoFlow [9]. However, we claim that the model is novel in terms of generality: (i) we employ relative bandwidth requirement, while CoFlow counts only flows, and (ii) we even do not restrict all the mathematical analyses to only network resource (it can be a mixture of CPU, memory and link bandwidth). For clarity we give a small motivating example and provide a formal definition.

An example. Imagine we have a tenant which has three virtual machines (VMs) — **A**, **B** and **C** — designated for some *job*. The job is processed as follows: VM **A** (*master node*) receives the job, splits it to two tasks, sends them to VMs **B** and **C** (*workers*) and collects the results. On a high level, the performance of the entire job is measured by the time VM **A** receives the last result. For simplicity assume that CPU time for tasks is the same for both workers or negligible in comparison to communication time (see Fig. 4).

We are interested in finding the amount of resource allocated to communication channels A–B and A–C. For a moment, imagine that after our allocation for the

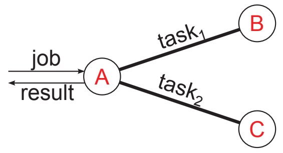


Figure 4: Task model

channel A–B the communication time is *10* ms and for the channel A–C it is *100* ms. Independently of the performance on the channel A–B, the entire task will not complete in time faster than 100 ms. In other words, this means that the performance of the task is determined by the slowest channel. Moreover, there is no benefit in receiving the excess bandwidth on a non-bottleneck channel A–B, which may be given to or traded with another tenant. However, we name A–C slowest channel in terms of completion time, in fact the flow rate on channel A–C can be 10 times of that on channel A–B. The fact of slower compeltion time comes from the fact that the job sends 100 times more data over A–C, than A–B.

The aforementioned problem can be solved if the tenant could explicitly proclaim how many bits per second she needs on channel A–B for every bit per second on channel A–C. This requirement leads us to the following definition of the task model.

Definition 1 (Task model). *A set of (communication) tasks of a tenant is defined by a set of virtual machines S and bandwidth requirements D for a set of links L over which the communication between these machines occurs. The job is said to be limited (after resource allocation) by the smallest ratio ($\frac{A_l}{D_l}$) between allocated (A_l) and demanded (D_l) bandwidth at some bottleneck link l (the slowest task in the set).*

In other words, this means that an increase in the allocated bandwidth $A_{l'}$ on some non-bottleneck link $l' \neq l$, will not improve the performance of the job, since equal proportionality between demanded and allocated bandwidths must be observed for all tasks (involved in the given job). In the example above the unnecessary capacity on the channel A–B can be given to some other tenant.

For simplicity, we limit ourselves to communication tasks. However, the limitation is not crucial. The model can be extended to a mixture of communication, computational and memory resource sharing. The latter requires some unified metric to make them comparable, for example, the time. For instance, the master node can send a task over a slow link to a fast machine, or over a fast link to a slow machine; the completion time defines what is preferred. Thus, having the same motivation for our work as Coflow [9], we have a different understanding of the resources than in Coflow.

3.2 Communication and demand model

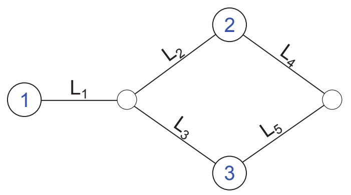
Let us define a graph (V, E), where V are nodes ($|V|= M$) and $E = \{L_i, i \in N\}$ are edges or links ($|E|= N$). Capacity of each link is limited by C_i ($|L_i|\leq C_i$). Any communication channel between two stations goes by some path and each path is defined by an ordered sequence of adjacent links (with no loops).

Consider K jobs are given, such that each job is executed by a single tenant, or a player. Each player, in turn, has a set of virtual machines or stations ($M^k, k \in K$) placed on the graph (each station is associated with some node). For simplicity, stations of each player are enumerated as $\{1, 2, \ldots, |M^k|\}$ (see Fig. 5a, where numbers show the stations placement). Additionally, each player has certain communication requirements for each pair of stations ($d^k_{i,j} \geq 0 \ \forall i, j \in \{1, 2, \ldots, |M^k|\}$ for $k \in K$) (Fig. 5b). We call matrix $d^k_{i,j}$ for tenant k *a demand matrix* (by analogy to demand vectors in DRF) and the values it has we consider to be relative for each k as in the task model.

Each tenant can produce a restricted mapping of own stations to the graph (reordering). The player has fixed physical machines to run the code on (see Fig. 5a, where numbers correspond to physical machines), but has some freedom to choose which code to run on the machines, e.g., what task to run on what VM (e.g., Fig. 5c and Fig. 5d for alternative placements). Practically a change of the role means that a tenant sets a permutation π of numbers $\{1, \ldots, |M^k|\}$, which corresponds to initial mapping to the graph. The whole set of permutations may be restricted to some feasible set Π, i.e. permutations that are not in this set are not allowed. Component i of permutation π means that the task i will be run on station $M^k_{\pi(i)}$ (instead of default M^k_i). We call permutation (or change of mapping) operation *a swapping*.

A single resource can be reused by multiple components of the demand matrix, for example in Fig. 5c the link L_1 is used twice. To compute the total (relative) requirement for a link we compute a *demand vector* (by analogy to DRF), which represents a flattened version of demand matrix: $D^k = \{D^k_1, \ldots, D^k_N\}$. Component D^k_i of a demand vector means how much bandwidth should be allocated on a link i if there is D^k_j of bandwidth on a link j. Fig. 5c shows the case when the tasks A, B, C are mapped to nodes 1, 2, 3 respectively. The demand vector for links $\{L_1, L_2, L_3, L_4, L_5\}$ is $\{2, 1, 1, 0, 0\}$ and Fig. 5d shows that the demand vector for mapping $\{2, 1, 3\}$ is $\{1, 1, 0, 1, 1\}$.

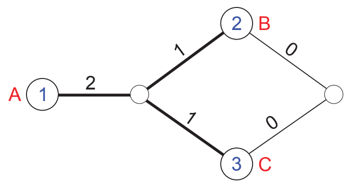
It can easily be seen that a tenant can manipulate network requirement by modifying (lying on) demand vectors (Strategy I) or by swapping between different permutations (Strategy II).



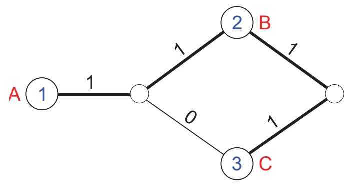
(a) Graph

	A	B	C
A	0	1	1
B	1	0	0
C	1	0	0

(b) Demand matrix



(c) Mapping: Case 1



(d) Mapping: Case 2

Figure 5: Graph, demand matrix and non-symmetric mapping cases: (a) The initial graph, where numbers represent nodes available for a tenant for placement; (b) The demand matrix in undirected form; the numbers are relative and are defined as in the task-model; (c) Case 1 placement, corresponds to mapping $A \to 1, B \to 2, C \to 3$; (d) Case 2 placement, corresponds to mapping $A \to 2, B \to 1, C \to 3$ or permutation $(2, 1, 3)$.

3.3 Utility function and the standard form of demand vectors

The task model was defined in relative values. To unify the understanding of performance and produce meaningful comparable results we introduce the following two definitions.

Definition 2. *In a standard form all demand vectors* $(D_1, \dots, D_N)$ *satisfy following condition* $0 \leq D_i \leq 1 \forall i$ *and* $\exists j : D_j = 1$

Definition 3. *Allocation utility of a player is defined as* $\alpha = \min_{i \in N : D_i > 0}\{\frac{A_i}{D_i}\}$, *where* A_i *is the allocation of resource* i *the player gets and* $D = (D_1, \dots, D_N)$ *is the demand vector in the standard form.*

It is easy to see that any non-zero positive demand vector $(D_1, \dots, D_N)$ can be written in a standard form as follows $(\frac{D_1}{\max_{\forall i} D_i}, \dots, \frac{D_N}{\max_{\forall i} D_i})$ and the utility is a multiplier of demand vector in a standard form which determines the cluster share of a tenant.

4. STEM ARCHITECTURE

We design a Strategy-proof Task-Enforcement Mechanism (STEM) using the notion of task-aware allocation in a form of demand vectors – a vector with a single numerical value for each link. The demand vectors are composed of the quantity of communication flows (single flow for each unique VM-to-VM pair[4]) over each link. As the quantity of flows are known to the network through hypervisors, there is no need for any API to expose the flow structure. By requiring the demand vectors to be defined as flow count we achieve that all flows of the same tenant obtain the same allocation. Moreover as a fairness metric, we equalize all flows of different tenants. In such a case, each flow at the network will be allocated with the same bandwidth.

This approach was constructed with a reference to DRF. The whole mechanism is similar to DRF for absolute values, where we count all the flows to produce a demand vector. Then we reduce it to a standard form demand vector. Finally, each demand vector is weighted with a value of maximum number of flows that each tenant has (the maximum over all links).

From the tenant perspective nothing was changed. It works as before. From the cloud perspective, hypervisor observes all the flows that originated locally. If a new flow is created or an old one is deleted, the hypervisor sends this information to a controller (Figure 9). Controller adds a new flow or deletes an old flow from the list. Using the list of flows, the controller can compute how many flows each link has. If link L has capacity C_L and observes D_L unique VM-to-VM flows (that is the value of a cumulative demand of all tenants for this link), then link L can allocate no more than $\frac{C_L}{D_L}$ to each flow. The controller computes the maximum achievable allocation the network can provide to all flows equally as $X = \min_L \left\{\frac{C_L}{D_L}\right\}$. The value X is the only thing that the controller needs to distribute to all hypervisors. Each hypervisor, having computed value X, can start to rate-limit all flows by this value.

All the computational mechanisms that we described above can be easily written in a recursive manner, and, thus, can easily scale. Although the controller needs to know whole network structure, we can split it into subcontrollers that know only a part of the network. For example, if a subcontroller A computes value X_A as described above and a subcontroller B computes value X_B, then we can say that the consensus value on allocated bandwidth is $\min\{X_A, X_B\}$.

We can extend the definition of the demand vector with a notion of partial flows. For some jobs there is no need to have all flows of the same size, some of the communication tasks are small. In that case instead of counting it as a full flow – adding a unit – we can add a fraction of a unit to the demand vectors' components, which are affected by this flow. That exactly is the place where CoFlow-like API can benefit by exposing the quantity of the "unity" that each flow needs. Note: at least one flow of each tenant should be non-fractional, otherwise the tenant simply gets less bandwidth allocation at all links than she could. The same way we can

[4]We will talk about multipath options later.

treat multipath flows which have the same originating and destination VMs: each of the flow receives proportional fraction of the unity (of a single flow).

The main drawback of the above-mentioned mechanism is the absence of work conservation property: the hypervisor uses strict rate-limiters for proposed allocations. To deal with this problem we introduce an extension mechanism – STEM+. For this mechanism we define two classes of traffic: guaranteed traffic (or high priority) and work-conserving traffic (low priority). STEM+ uses the same mechanism as STEM to compute allocations, however these bandwidth allocations form guaranteed traffic. The excess of the guaranteed traffic that VMs try to send is work-conserving traffic. There is almost no changes to the hypervisor compared to STEM, except that right now the hypervisor does marking of the packets instead of rate-limiting. All traffic that is bellow a given rate is marked as priority traffic, all excess traffic remains unmarked.

The network switches should be able to differentiate two classes of traffic: marked and unmarked. While unmarked traffic can easily be dropped due to congestion, marked traffic should always be delivered and the STEM controller computes the rates which should be enough to satisfy that. The prioritization can be done with many legacy technologies, e.g. DiffServ can easily differentiate two classes of traffic (rather common assumption in the field), however an exact form of realization depends on network infrastructure hardware. In case the switches do not support any prioritization mechanism, the prioritization can be done at hypervisors in the same manner as ElasticSwitch [22] does it.

As it was mentioned, work-conservation comes with the cost of strategy-proofness. However, in this case work-conserving traffic is a second-class traffic; if the tenant values bandwidth traffic guarantees then we can say that we keep strategy-proofness at the level of guaranteed traffic. One additional benefit of STEM+ is that a new flow can start to work immediately as it created as work-conserving traffic. The delayed allocation will just move part (or whole) work-conserving traffic to the higher class of guaranteed traffic once new X is acquired. This mechanism is also more fault-tolerant as in the case of controller problem, all the traffic start to be legacy work-conserving traffic.

5. COMPARISON OF ALLOCATION MECHANISMS

The defined model allows a weakening of constrains if we consider that players can work directly with demand vectors instead of demand matrix. We think that the proceeding analysis can be generalized, but for the sake of simplicity and due to space limits we omit such generalization. With this assumption we are oblivious to the underlying network structure and VM placement model; instead users can directly manipulate demand vectors using Strategy I – lie about demand vector components – and Strategy II – change the order of demand vectors. As before the demand vectors are N-dimensional vectors, each dimension corresponds to a single link. In such setup we compare main allocation policies.

5.1 Pricing and Competitive Equilibrium from Equal Incomes (CEEI)

CEEI is a fundamental resource allocation mechanism. It was shown that any market-pricing scheme for multiple resources leads to CEEI allocation [24, 27]. Our simulation showed the same result for linear pricing and hyperbolic pricing which depends on the links loads. In both cases the pricing mechanisms always converge to CEEI, thus pricing mechanisms have similar property to those of CEEI, except that CEEI allows to find the allocations immediately while pricing mechanisms need some time to converge.

It means that given an equal budget to multiple players and a market-driven prices, which are based on resource load, the players end up allocating their budget with the same scheme as CEEI allocation policy provides. Thus, CEEI is a very important allocation policy as it covers a big class of pricing schemes for individual resources.

For two players the CEEI mechanism is defined as following

$$\begin{aligned}&\max(x \cdot y)\\ &\qquad \text{subject to}\\ &\quad a_1 \cdot x + b_1 \cdot y \le C_1 \quad \text{(first link)}\\ &\quad a_2 \cdot x + b_2 \cdot y \le C_2 \quad \text{(second link)},\end{aligned}$$

where (a_1, a_2) and (b_1, b_2) are demand vectors of the first and second player correspondingly.

In [13] the authors mention that CEEI is not strategy-proof for non-fungible[5] resources. It is easy to show that CEEI is not strategy-proof either when resources are fungible. However, we omit the proof of this obvious result due to space limitations. Instead, we focus on discussion of existence of Nash equilibrium (NE) for CEEI game with two players and two fungible resources. As we shall see, this equilibrium either is trivial (but unstable) or is bad as each tenant requests the same allocation of each resource independently of real demands, leading to equal share $1/K$ of all resources.

THEOREM 1 (EXISTENCE OF NE FOR CEEI). *For a 2-player CEEI allocation game NE exists and is undesirable.*

[5]Fungible resources can be substituted one by another without any drawback, e.g., two different paths from the same initial VM to the same destination VM are fungible resources; while CPU and memory are non-fungible resources – it is impossible to completely substitute one with another.

PROOF. Consider two resources of equal capacities ($C_1 = C_2$). Our solution can be generalized to non-equal capacities. Let demand vectors of two player be given in a standard form: $(1, a)$ and $(1, b)$. The case when $a = 0$ and $b = 0$ is trivial – it leads to $(1, 0)$ and $(0, 1)$ solution – however, it is also not stable – a slight change in the second component breaks this equilibrium. We assume that, $0 < a \leq 1$ and $0 < b \leq 1$.

If players choose vectors $(1, a)$ and $(1, b)$, then each will get $\frac{1}{2}$ of the resources (the same happens if $a = 1$ or $b = 1$). In this setting, both players will prefer to swap vectors, such that both can gain more than $\frac{1}{2}$. If both players choose vectors $(1, a)$ and $(b, 1)$ then the solution has the following form:

$$\begin{cases} x = \frac{1}{2}, & y = \frac{1}{2 \cdot b}, & \text{if } 1 \leq b \cdot (2 - a) \\ x = \frac{1}{2 \cdot a}, & y = \frac{1}{2}, & \text{else if } 1 \leq a \cdot (2 - b) \\ x = \frac{1-b}{1-a \cdot b}, & y = \frac{1-a}{1-a \cdot b} & \text{otherwise} \end{cases}$$

Now, for each point (a,b), $0 < a < 1$ and $0 < b < 1$ in CEEI it is beneficial for players to move their non-dominant components. The best response for any point (a, b) for the first player will be from a to $a' = 1/(2-b)$, while for the second player will be from b to $b' = 1/(2-a')$. It turns out, that this iterative process converges to $(1, 1)$. Hence, the sought NE is of kind when one or both of the players choose $(1, 1)$ as their demand vectors. □

This equilibrium has a negative effect on the system, because resources are utilized inefficiently: every tenant has a tendency to ask for resources which essentially she will never end up using. Moreover, it is beneficial for tenants to create a competition (and thus increase prices) on the links they do not need in reality. Investing higher portion of own budget into a congested link affects another player and leads to a better allocation.

The analysis is also extensible to N-dimensional demand vectors. We do not have a proof of this result for K players, but our numerical analysis shows that for any initial setup we had players always wanted to change their demand. Thus, we conclude that CEEI is not strategy-proof in general, i.e. generally for any two players the aforementioned misbehavior exist.

5.2 Task-enforcement allocation (STEM)

STEM utilizes this type of allocation by forcing each flow to achieve equal allocation X. STEM+ uses it only for the guaranteed bandwidth. As in this allocation type each flow is treated equally, the creation of extra blocking traffic (as shown in Figure 2) becomes non-beneficial for a tenant. The player knows that all tenants will get the same allocation X for each full flow, thus misbehaving against another tenant, the player decreases own allocation. At the same time, changing the roles of VMs as a strategy is beneficial to everyone if it is beneficial for the player, because it helps to lighten bottlenecked links. Finally, splitting jobs in space or in time becomes either beneficial for everyone by shifting bottleneck or it does not produce any improvement as all sub-jobs will get a fraction of the original allocation.

5.3 Other allocations

With the presence of work-conservation the scheme presented in Figure 2 is always viable. Because tenant B have a strong connection in utilization of both links L_1 and L_2. Flow $B_1 \to B_2$ cannot be bottlenecked at link L_1 and use work-conservation at link L_2. At another hand tenant A always has work-conservation property available at the link L_2.

However, we should note that the strategy-proofness is not broken only in the case of work-conservation. If there is no work-conservation property in the allocation, the Figure 2 is still a valid example. Tenant A can still benefit in $A_1 \to A_2$ creating jamming traffic at $A_3 \to A_4$. Although, there is no straightforward allocation change in $A_1 \to A_2$ by the jamming another link, the tenant B may decide to switch from the current path to another less congested path for $B_1 \to B_2$ communication. A new allocation after this iteration becomes again beneficial for player A. Obviously, the network without work-conservation can be highly underutilized.

From the above discussion follows that hose-model based allocation cannot be fully strategy-proof, except trivial cases.

5.4 Properties of STEM

DRF. As was said before STEM can be expressed in a from of dominant-resource fairness (DRF) [13], thus the following discussion applies to STEM. The goal of DRF is to equalize the percentages of dominant resources of participants. The usage of percentage makes some resources more valuable than others in the task model. To cope with this problem instead of equalizing the dominant share in percentage values we equalize them in absolute values. We use term *DRF-A* for DRF in absolute values and *DRF-P* for DRF in percentages.

LEMMA 1. *If all resources have equal capacities, i.e., $C_1 = C_2 = \cdots = C_N$, then an allocation defined by DRF-P is equivalent to DRF-A allocation.*

The allocation finding problem for DRF-A with demand vectors in a standard form reduces to a very simple formula.

DEFINITION 4. *Solution of DRF-A for demand vectors (D) in the standard form is defined by the following linear system*

$$\max\{x\},$$

subject to

$$X \leq \frac{C_j}{\sum_i D_j^i}, \forall j \qquad (1)$$

$$X = x_1 = x_2 = \dots x_M$$

or equally $X = \min_j\{\frac{C}{\sum_i D_j^i}\}$, where tenant i receives allocation $(X \cdot D_1^i, \dots, X \cdot D_N^i)$ and X is the same utility which every tenant gets.

Now, we will formulate the existence theorem on NE for DRF-A.

THEOREM 2 (EXISTENCE OF NE FOR DRF-A). *NE for DRF-A always exists.*

PROOF. DRF-A is defined for demand vectors in a standard form equalizes utility functions of the above game, *i.e.*, $X = x_1 = x_2 = \dots = x_M$. Thus, any selfish improvement of utility for any player increases utilities of all other players.

For NE to exist we need to prove that there is no endless process of utility increase. For that consider a total number of states in which the game can be at any moment: each player can be in one of $N!$ demand vector permutations and there are M such players, thus, it is $(N!)^M$. This number is bounded and there is no endless increase of utility values. □

Structure of NE. In DRF-A tenants choose own strategy based on the cumulative strategies of the others. There are multiple situations when different strategies (non-symmetric) produce the same result for the player. For example if only one resource is a bottleneck for every tenant for any strategy, then they will use the least utilization for this link, and independently of VM roles permutation for other links, this link is the limiting factor. Thus, there are multiple NE, which however produce the same utility value. With this we proved that these are weak NE.

We also see, following the proof on existence, that there is a possibility of multiple different NE. For that consider the smallest example. Let us have three players sharing three resources: player A with demand vector $(1, 0.8, 0.3)$, player B – $(1, 0.7, 0.5)$ and player C – $(1, 0.5, 0.1)$. We find that with such setup there are 3 different NE, utilities:

1. $\alpha = \frac{1}{2.1}$ for A: $(1, 0.8, 0.3)$, B: $(1, 0.7, 0.5)$, C: $(1, 0.5, 0.1)$;
2. $\alpha = \frac{1}{2.0}$ (optimal) for A: $(1, 0.8, 0.3)$, B: $(0.5, 1, 0.7)$, C: $(0.5, 0.1, 1)$;
3. $\alpha = \frac{1}{2.3}$ (pessimal) for A: $(1, 0.8, 0.3)$, B: $(0.7, 0.5, 1)$, C: $(0.1, 1, 0.5)$.

The differences between Nash equilibria in this example are minimal, however we clearly see that there exists an optimal NE, which is called Pareto-efficient, as well as we have a pessimal NE.

Approximate algorithm to find the optimal NE. The main property of DRF-A is that the utilities are equalized, thus a search for NE reduces to an optimization problem. From the proof of the Theorem on existence we know that the best response strategy will lead to a NE.

As an approximation we apply the following partial best response once per each player. Assume that i players have done their turn, now the best response for $i+1$st player is to reduce the sum of demand vector components on each link, for that we sum up all demand vectors of previous i players into a single cumulative vector $(t_1, t_2, \dots, t_N)$. The best response of player $i+1$ to that vector is to swap the minimum value of the player demand vector to the maximum value of cumulative vector, after that the second minimal value to the second maximal value of cumulative vector, and so on.

Let us study how well this approximation works. The following lemma will help us.

LEMMA 2. *For demand vectors in the standard form, in each step of the partial best response algorithm the difference between the maximum and the minimum value of the cumulative vector is no more than one.*

PROOF. Straightforward. □

THEOREM 3. *The approximation algorithm achieves utility not worse than $1 + \frac{N}{S}$ times less compared to the best possible (idealistic) solution, where $S = \sum_{\forall i,k} D_i^k$.*

PROOF. Let us name the final cumulative vector as $(t_1, t_2, \dots, t_N)$ and the sum of all components be equal to $S : S = \sum_i t_i$. Let t_{max} and t_{min} be respectively the maximum and the minimum components of the cumulative vector. From the lemma we know that $t_{min} + 1 \geq t_{max}$. We also know that $S = \sum_i t_i \geq N \cdot t_{min}$. Finally, $t_{max} \leq \frac{C}{N} + 1$ and utility is $\alpha \geq \frac{N}{C+N}$. Now we need to compare it with the optimal utility which is achieved in Pareto-efficient NE, let us name it as α^*. We know that for given demand vectors the best and ideal solution is achieved when all links share the same cumulative demand, i.e. $(t_1, t_2, \dots, t_N) = (\frac{C}{N}, \dots, \frac{C}{N})$. This ideal situation rarely achievable in practice provides us the upper bound on α^*: $\alpha^* \leq \frac{N}{C}$.

Finally, we get that $\frac{\alpha^*}{\alpha} \leq 1 + \frac{N}{S}$.

□

We know that each vector has at least one unit component, thus $S >= N$. Thus, in extreme our formula shows that the ideal utility is not better than twice higher than our approximation for any number of players. In practice if $S = N$, our algorithm achieves the ideal utilization. In case $S >> N$ the algorithm produces the result very close to ideal.

For an example above (player A with demand vector $(1, 0.8, 0.3)$, player B – $(1, 0.7, 0.5)$ and player C – $(1, 0.5, 0.1)$) $S = 5.8$ while $N = 3$, thus by theorem $\frac{\alpha^*}{\alpha} \leq 1.517$. The best NE produces utility 0.5. If we run the approximation algorithm above step by step, then on the first step we will take the demand vector A in any order, say $(1, 0.8, 0.3)$, on the second step the best response of player B for player A will be a demand vector $(0.5, 0.7, 1)$. Combining demand vectors A and B we get a partial cumulative vector $(1.5, 1.5, 1.3)$, finally the best response of player C for that will be demand vector $(0.1, 0.5, 1)$ which produces the final cumulative vector $(1.6, 2, 2.3)$; the maximum component of which is 2.3. Finally, the utility this algorithm gets is $\frac{1}{2.3} \approx 0.435$. The ratio of the ideal to the approximation is $\frac{\alpha^*}{\alpha} \approx 1.19$ which is by the previous theorem is bounded from above by 1.517.

5.5 Analytical results

The results we have are summarized in Table 1. The first point to note is that the allocations the mechanisms have are final for all except pricing mechanisms. For pricing mechanism the allocation needs some time to converge, i.e., a tenant has to find how much to pay for each link. The strategic property is good only for DRF: STEM has sustainability to lie about demands. Performance is bad for CEEI as the tenants are unwilling to share the cluster. Each gets $1/K$ of resources and due to the task model a lot of own resources remain unused, while could be traded with others. The same holds for local allocation policies with allocation enforcement. However, the reason is that local mechanisms cannot efficiently distribute resources using local knowledge. Work conserving property helps local allocation policies to get rid of this but makes the strategic properties of the allocation worse.

STEM inherits from DRF the property of work-conservation on task level, i.e. there is no allocation that will increase the performance of anyone without harm for others, but there is no work-conservation at the flow level. The strategic property of STEM also leads to a Pareto-efficient solution, which is beneficial for everyone. Finally, fairness is achieved on STEM by the task-enforcement method itself, while CEEI fairness is due to full isolation of tenants. Any local allocation policy cannot achieve fairness at the global level. They can achieve local fairness (FairCloud aims to do so), however bad strategic property can lead the tenants to another unfair allocation after re-allocation and misbehavior.

6. TRACE-DRIVEN SIMULATION

So far we investigated the strategic properties of STEM, STEM+, CEEI, pricing and task-unaware allocations theoretically. Many of the properties are qualitative thus we cannot easily measure them quantitatively. Now we would like to measure performance and fairness characteristics of STEM and STEM+, rather than strategic properties of the protocols. For that we choose a trace-driven approach. We use the traces collected from Facebook production cluster running *MapReduce* jobs. To characterize these traces, we found that the cluster was comprising about 3200 physical nodes. Furthermore, for such a relatively large cluster, the mean job arrival/completion rate was 63 *jobs/second*, whereas the minimum value of this number was 4 and the maximum was 1147. We also found that the average, minimum and maximum number of active tenants at any given time instant was 44, 32 and 63 respectively.

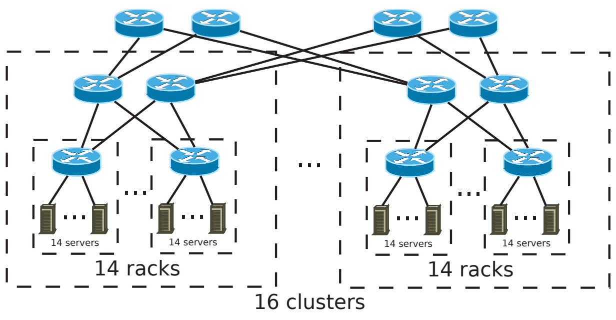


Figure 6: Simulated datacenter topology

Because Facebook traces did not contain topology information we had to infer it. The FairCloud work [20] assumed a plain tree topology which is not right. As [5] study suggests, industrial datacenters usually have canonical 3-Tier Cisco architecture. We assumed the network to have topology as shown in Fig. 6 which is similar to standard Cisco campus network[6]. It is 3-Tier architecture with 3136 servers and 260 routers. All links have capacity of $1000 Mbps$.

Following the work described in FairCloud paper [20], we considered one-hour windows in the trace and observed the number of jobs involved in active shuffle at a minute's interval in those hours. Then we chose time instants at which there were most jobs in active shuffle. We then created snapshots and computed corresponding traffic matrices, which we further used to observe the bandwidth allocations with different policies on the 4× oversubscribed topology.

We simulated STEM and STEM+ and benchmark them against state-of-the-art fair allocation algorithms for the cloud, PS-L and PS-N. We could use other al-

[6]http://www.cisco.com/en/US/docs/solutions/Enterprise/Campus/campover.html

Table 1: Comparison of results

Allocation type	Allocation result	Strategic property	Aggregate performance	Task-level performance	Fairness
Pricing	*bad*	*bad*	*bad*	*bad*	*good*
CEEI	*good*	*bad*	*bad*	*bad*	*good*
w/o work-conservation	*good*	*bad*	*bad*	*bad*	*NA*
w/ work-conservation	*good*	*bad*	*good*	*bad*	*NA*
STEM	*good*	*good*	*bad*	*good*	*global*
STEM+	*good*	*good**	*good*	*good*	*global*

gorithms to compare STEM and STEM+ against, however, as those algorithms were not aiming at fairness they will be in disadvantageous position.

First of all, we compare how STEM and STEM+ treat different classes of tenants. From Figure 7a and Figure 7b we observe that both mechanisms treat tenant classes identically. While their comparison against PS-L and PS-N shows that STEM and STEM+ provide bandwidth guarantees for slowest flows, while PS-L and PS-N cannot do that (Figure 7c). We also observe that STEM+ performance for the fast flows is close to PS-L and PS-N, while STEM by design keeps the same value for all flows. This effect comes from work conservation property. With this property all allocation algorithms will achieve similar performance, but STEM+ also efficiently trades excess in fast flows performance for the increase in slow flows performance. Figure 7d validates our assumption that work conservation property achieves the same utilization of the network independently of the allocation algorithm: all three mechanisms (PS-L, PS-N and STEM+) conserving allocation achieve almost identical aggregated bandwidth. On another hand STEM cannot achieve such high aggregated bandwidth utilization without work-conservation; however this is compensated by the benefit of strong strategy-proofness. If all tenants in the network are task-aware and provided exact task allocation model, then they simply cannot benefit from the aggregated bandwidth which STEM+ provides. They will still wait for the same amount of time for the slowest links. STEM+ however makes it easy to provide not exact task model to the cloud (if the tenant does not know exact values beforehand), then work-conservation can benefit at the links where provision of the traffic requirement was under-estimated.

One of the discussion points that remains is how fair is it to provide so much network usage to big tenants. Although, they stop to be incentivized to misbehave, having M mappers and R reducers they can obtain O(M*R) network usage, while payment to cloud provider is proportional to O(M+R)[7]. However, we observe that it is not true, their influence on the network and requirement of the network is not quadratic as it may seem at the first glance but rather linear to their size (Figure 8). Thus the network usage of big tenants is proportional to the number of VMs they use.

[7]Assuming, that cloud providers take flat payments per VM.

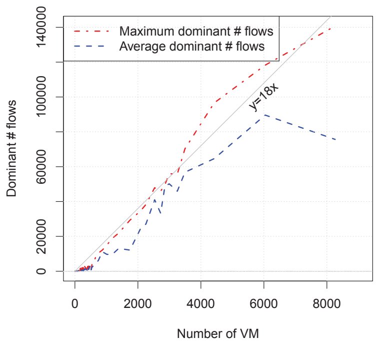


Figure 8: Maximum demand

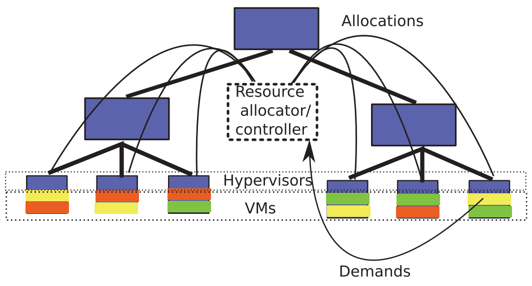
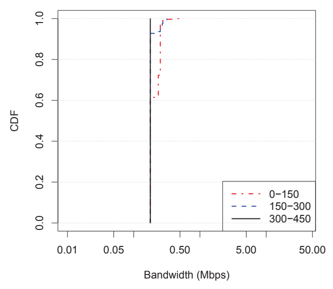


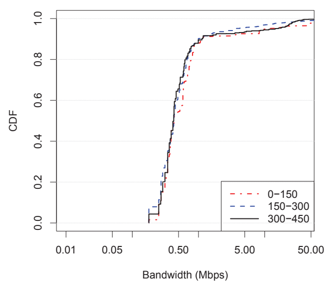
Figure 9: Resource allocation architecture

7. PRACTICAL CONSIDERATIONS

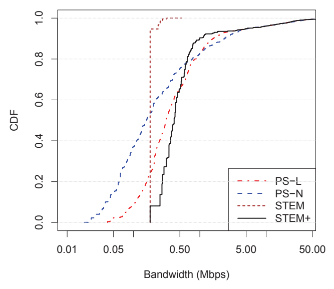
Feasibility. STEM/STEM+ architecture shown in Figure 9. We envision that all physical servers are provisioned with a hypervisor and that a centralized controller is present in the network. The role of hypervisors is to count the #flows and produce rate-limiting or marking functionality. These hypervisor roles are often met in corresponding literature [21]. As we normally observe oversubscription, the functionality can be implemented deeper in the network (closer to the bottleneck), e.g., a single exit point of a rack can do the



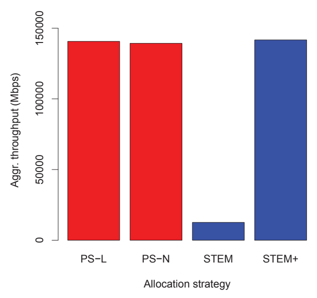
(a) STEM fairness



(b) STEM+ fairness



(c) Performance of allocation mechanisms



(d) Aggregated bandwidth

Figure 7: Simulation results

same job instead of hypervisors. The controller can be implemented in scalable distributed manner, even with single controller the solution is feasible and similar approaches met in OpenFlow/SDN literature. STEM+ requires two-class priority differentiation, which is also a common assumption about the network. Moreover STEM+ allows delayed fault-tolerant calculation of the allocations.

In future (we already allow it), the tenants can express their demand vectors explicitly to the infrastructure. We have benchmarked the performance of a single controller placed in the network comprising 3200 servers. From Facebook traces we inferred job arrival and departure rates, which gave us a rough estimate on how frequently a centralized controller will trigger calculations to update tenants' allocations. We further assumed that once allocations are calculated, a centralized controller can reliably convey new allocations to all hypervisors via a single broadcast message containing bandwidth allocations for all tenants and their VMs.

We show a summary of these experimental results in Table 2. Our suboptimal implementation of controller could calculate allocations 1260 times per second which is enough for the worst case for a given datacenter. Note that calculations can be easily scaled up by adding more controllers, each responsible for some part of the network. Each controller can calculate possible allocations for all tenants using only data for some part of the network. Then calculated allocations can be easily combined by applying the component-wise minimum.

The data sent by controller is quite small as in DRF only one number per tenant must be broadcasted. We estimate that 10 bytes per tenant is enough (4-byte tenant identity, 4-byte allocated bandwidth and 2 bytes reserved). Given that the average number of tenants in dataset is 48, broadcast messages from a controller will require bandwidth of 230Kb/s on average, and 4.4Mb/s at the worst case. In case of high-bandwidth links in datacenters, such amount of control traffic is considered feasible.

Table 2: The number of jobs started and completed

median	mean	95%	max
28	60	253	1248

Multipathing. We used both fat-tree and standard-tree topologies with oversubscriptions in our simulations. However, we report results only for fat-tree. Fat-tree allows to have multiple paths. We used a few heuristics to improve allocation by utilization of multiple paths. We find that ECMP already distributes the load uniformly among different paths, generally we could achieve improvements in allocations, but no more than 10%. In our opinion, it is a rather small number for the computational costs. At another hand, STEM allows to use multiple paths without any changes. E.g., one flow with count 1 can be split into two flows with different paths and counts as $\frac{1}{2}$ each.

Admission control and placement. We consider admission control and placement orthogonal to our approach. The cloud provider should itself calculate how many tenants can be satisfied with given network and usage expectations. What STEM can provide is the ability to equalize all flows. If a tenant experiences worse network performance than the expected, it means that the problem is in the network provision. VM placement is an NP-hard problem, providers normally allocate VMs by using locality as a heuristic. Smart tenants in some cases can improve placement, however generally that should not be expected from them. In the simulations we haven't done any replacement based on local experience after an initial placement.

8. RELATED WORK

Recently, bandwidth allocation in data centers received much attention from researchers. In production clusters, Hadoop Fair Scheduler [1] allocates resources using fixed-size shares called slots. Various reservation-based schemes including Oktopus [3], Gatekeeper [23] and SecondNet [14] were proposed for bandwidth guarantees without work conservation properties. Those often implement a hose model [12] where each user is connected by a virtual link with minimum guaranteed bandwidth. Oktopus enabled a symbiotic relationship between tenants and datacenter providers by defining an abstraction model for requesting a virtual network allocation. Using the model, a customer can choose either more expensive virtual cluster with guaranteed bandwidth among all VMs or more economic virtual oversubscribed cluster that places nodes in several groups with limited bandwidth.

FairCloud [20] analyzed the tradeoffs between payment proportionality, high utilization and minimum bandwidth guarantees. The authors proposed three allocation policies, including PS-L for proportional sharing on the link-level, PS-N on the network-level, and PS-P taking into account link proximity. While FairCloud policies can be efficiently implemented in switches, they lack the task concept which is necessary to capture the dependency among allocations on links forming a common path.

Seawall [25] allocates network bandwidth in datacenters based on policies specified by administrators. It utilizes tunnels between hypervisors to control congestion for high scalability in the absence of fixed resource reservations.

Kumar et al [17] consider network allocations in production clusters where tenants are cooperating. They propose to share resources to improve overall performance of tasks to help parallelizing the applications with less focus on fairness.

Mogul and Popa [19] provide a survey of network allocation mechanisms in cloud computing. They argue that allocation fairness is not a concern for tenants, but isolation and pricing transparency is. They suggest the idea of discriminate pricing for datacenter resources, where each tenant has to pay a different amount for same level of service. They point out that bandwidth guarantees can be provided on varying level of granularity, starting from per-tenant aggregate, to per-VM hose model, a pipe model between each pair of VMs, and per TCP flow between VMs.

Orchestra [10] was proposed as a management framework to control data transfers in datacenters. It defines a scheduling model permitting prioritization of transfers and improves performance of common operations such as data broadcast or shuffle. Using Cornet, a custom version of BitTorrent protocol, Orchestra can reduce the completion times of broadcast tasks and improve the performance of high-priority transfers. Orchestra defines a transfer as a combination of all network flows through a multi-stage job and goes beyond prior work on flow-level scheduling.

Ballani et al [4] argue based on datacenter measurements that traffic between tenants can be as high as 30% of overall traffic within the datacenter. They introduce Hadrian, a datacenter manager that controls VM placement within the datacenter to provide minimal bandwidth guarantee and take into account a pattern of inter-provider traffic.

Coflow [9] provides a networking abstraction API to communicate application requirements to a cluster computing platform. Coflow combines several flows between two groups of machines that work to accomplish a common goal, such as minimize completion time or meet a deadline. The proposed API enables creating, updating or terminating coflows.

Xie et al [29] argue that previous works on bandwidth allocation in datacenters ignore the fact that the task demand can vary a lot during task execution lifetime. This is supported by profiling measurements of several popular cloud applications such as a word count or Hive. They propose Proteus, a system that allocates bandwidth to tasks based on their network usage profile. This reduced the costs to tenants and improves datacenter utilization, compared to bandwidth reservation for entire task duration.

Webb et al [28] suggest to select a path within the datacenter network based on requirements of a job. Possible criteria include high bandwidth, flow isolation or resilience. The study compares the efficiency of path selector compared to currently popular VLAN+ECMP approach.

Overall, fair bandwidth allocation is a well studied topic in computer networks [6]. Researchers proposed solutions typically emulating max-min allocations to tenants treating bandwidth as the same interchangeable resource. Ballani et al. propose a simple pricing scheme for location independent tenants [2].

The recent work on task-aware allocation Baraat [11] shows that task-aware allocation can be implemented by producing relative ordering over all flows of different tasks. If flows of one task are of a higher priority than flows of another task, then the first job will be completed faster.

Attempts to generalize DRF to balance between fairness and efficiency resulted to mechanisms that are not strategy-proof [16].

Sinbad [7] proposes to optimize performance of Cluster File System (CFS) by replica placements during write operations. By avoiding congested links, the job completion time is reduced without negatively affecting the balance of storage.

A whole separate area of related work is virtual network embedding [8, 30]. This problem is present, for example, in Emulab testbed where a set of experiments with defined CPU and bandwidth requirements need to be mapped to a physical substrate network of servers.

The task requirements are generally assumed to be known for this problem, whereas for MapReduce tasks the Reduce phase resources cannot be fixed until the Map phase completes. The main difference of virtual network embedding from our work is that VNE does not consider a notion of fairness in resource allocation. A task is either granted its requests or denied due to access control.

A concept of ElasticSwitch [22] guarantees bandwidth allocation and work conservation to tenants simultaneously, while not demanding any support from network switches. CloudMirror [18] takes into account application requirements when allocating bandwidth in the cloud.

Predictable Shared Cloud Storage (Pisces) [26] applied DRF to ensure fairness and performance guarantees between tenants in a public cloud. However, it assumes well-provisioned network without oversubsciption and hence does not address network abuse of by the tenants.

9. CONCLUSION

In this work we introduced a novel strategy-proof task-aware allocation mechanism, namely STEM. It has a simple architecture and at the same time strong properties for tenant behavior. No any other allocation type can provide necessary properties. Moreover, we show that the pricing mechanism is not strategy-proof and in some cases can break the sharing incentive for the tenants. To our knowledge this is also the first work which considers strategic tenants. The extension of the protocol, STEM+ achieves many of the important properties, as well as shows the same aggregate bandwidth performance as other work-conserving allocation methods.

10. REFERENCES

[1] Hadoop fair scheduler. http://hadoop.apache.org/docs/r1.1.1/fair_scheduler.html.

[2] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. The price is right: towards location-independent costs in datacenters. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 23:1–23:6. ACM, 2011.

[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 242–253, New York, NY, USA, 2011. ACM.

[4] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea. Chatty tenants and the cloud network sharing problem. In *NSDI'13*, Apr. 2013.

[5] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

[6] J. M. Blanquer and B. Özden. Fair queuing for aggregated multiple links. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 189–197, New York, NY, USA, 2001. ACM.

[7] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proc. of ACM SIGCOMM'13*, 2013.

[8] M. Chowdhury, M. R. Rahman, and R. Boutaba. Vineyard: virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Trans. Netw.*, 20(1):206–219, Feb. 2012.

[9] M. Chowdhury and I. Stoica. Coflow: a networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 31–36, New York, NY, USA, 2012. ACM.

[10] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM'11, pages 98–109, New York, NY, USA, 2011. ACM.

[11] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks, September 2013.

[12] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 95–108, New York, NY, USA, 1999. ACM.

[13] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant resource fairness: fair allocation of multiple resource types. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 24–24. USENIX Association, 2011.

[14] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, Co-NEXT'10, pages 15:1–15:12, New York, NY, USA, 2010. ACM.

[15] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM*

SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '12, pages 127–138, New York, NY, USA, 2012. ACM.

[16] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM'12*, pages 1206–1214, 2012.

[17] G. Kumar, M. Chowdhury, S. Ratnasamy, and I. Stoica. A case for performance-centric network allocation. In *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, HotCloud'12, Berkeley, CA, USA, 2012. USENIX Association.

[18] J. Lee, M. Lee, L. Popa, Y. Turner, P. Sharma, and B. Stephenson. Cloudmirror: Application-aware bandwidth reservations in the cloud. In *ACM HotCloud'13*, 2013.

[19] J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *SIGCOMM Comput. Commun. Rev.*, 42(5):44–48, Sept. 2012.

[20] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: sharing the network in cloud computing. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 187–198, New York, NY, USA, 2012. ACM.

[21] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 351–362, New York, NY, USA, 2013. ACM.

[22] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM'13*, 2013.

[23] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes. Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proceedings of the 3rd conference on I/O virtualization*, WIOV'11, Berkeley, CA, USA, 2011. USENIX Association.

[24] W. Shafer and H. F. Sonnenschein. Market demand and excess demand functions. In K. J. Arrow and M. Intriligator, editors, *Handbook of Mathematical Economics*, volume 2, chapter 14, pages 671–693. Elsevier, 4 edition, 1993.

[25] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 23–23, Berkeley, CA, USA, 2011. USENIX Association.

[26] D. Shue, M. J. Freedman, and A. Shaikh. Performance isolation and fairness for multi-tenant cloud storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 349–362, Berkeley, CA, USA, 2012. USENIX Association.

[27] H. R. Varian. Equity, envy, and efficiency. *Journal of Economic Theory*, 9(1):63–91, 1974.

[28] K. C. Webb, A. C. Snoeren, and K. Yocum. Topology switching for data center networks. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.

[29] D. Xie, N. Ding, Y. C. Hu, and R. Kompella. The only constant is change: incorporating time-varying network reservations in data centers. *SIGCOMM Comput. Commun. Rev.*, 42(4):199–210, Aug. 2012.

[30] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, Mar. 2008.