Packer: Minimizing Multi-Resource Fragmentation and Performance Interference in Datacenters

Daniel S. Marcon*†, Marinho P. Barcellos†
*University of Vale do Rio dos Sinos (UNISINOS), RS, Brazil
†Federal University of Rio Grande do Sul (UFRGS), RS, Brazil
Email: {daniel.stefani, marinho}@inf.ufrgs.br

***Abstract*—Cloud applications perform rich and complex tasks, with time-varying demands for multiple types of resources (CPU, memory, disk I/O and network). However, multi-resource allocation is APX-Hard and, consequently, providers simplify it by (i) allocating computing resources according to slots (which leads to fragmentation); and (ii) allowing the network to be shared in a best-effort manner (which leads to performance interference among applications). Recent efforts cannot minimize multi-resource fragmentation and, at the same time, provide guaranteed network performance. In this paper, we introduce Packer, a scheme that aims at minimizing multi-resource fragmentation and providing predictable and guaranteed network performance with work-conservation. Packer employs a novel allocation strategy that (a) extends previous heuristics developed for multi-dimensional bin packing; and (b) uses as input a new abstraction, called Time-Interleaved Multi-Resource Abstraction (TI-MRA), for specifying temporal multi-resource requirements of applications. It also leverages Software-Defined Networking to dynamically enforce bandwidth guarantees and to provide work-conserving sharing. Since Packer brings more benefits if temporal requirements are specified, it is better suited for applications that present a predefined behavior, repeatedly running the same type of tasks with similar input sizes and data sets (such as PageRank and traffic analysis). Results show that, in comparison to the state-of-the-art, acceptance ratio is increased, datacenter utilization is improved (i.e., fragmentation is minimized), provider revenue is augmented and applications achieve predictable and guaranteed network performance with work-conservation, with the cost of taking more (yet acceptable) time to allocate applications.**

I. Introduction

Cloud applications often perform rich and complex tasks, with different temporal demands for multiple types of resources (CPU, memory, disk I/O and network) [1]. They typically have multiple stages, where a subsequent stage can only start after the previous stage finishes [2]. Consequently, any resource that becomes a bottleneck and delays a stage may slow down the entire application.

In this context, multi-resource allocation is a key building block of cloud datacenters. Because the allocation problem is APX-Hard [3], state-of-the-art proposals typically simplify it by (i) allocating computing resources according to slots [1]; and (ii) allowing the network to be shared in a best-effort manner, which leads to performance interference among applications [4], [5]. The former (slot-based allocation) usually causes over-allocation, leading to wastage (as applications do not use all of their allocated resources) and fragmentation (the sum of available resources would be enough to accept an incoming application, but these available resources are scattered throughout the infrastructure) [6]. In particular, fragmentation results in less applications being accepted in the infrastructure and in lower datacenter utilization. The latter (performance interference) results in poor and unpredictable network performance for applications.

ISBN 978-3-901882-94-4

Allocating resources to applications in datacenters has been the main focus of several recent efforts [1], [4], [6]. Tetris [6] considers multiple resources at allocation time, but does not provide bandwidth guarantees along the entire network. Dominant Resource Fairness (DRF) [1] also considers multiple resources, but focuses on fairness (which may result in fragmentation [6]). Silo [4] offers predictable network performance, but allocates computing resources according to slots. In general, these approaches cannot minimize multi-resource fragmentation and, at the same time, provide guaranteed network performance.

In this paper, we propose Packer, a scheme for large-scale Infrastructure-as-a-Service cloud datacenters. Its design is based on two observations: (i) applications have complementary demands across time for multiple resources [6]; and (ii) utilization of different resources peaks at different times [7]. Packer has two objectives: minimizing multi-resource fragmentation (consequently, increasing datacenter utilization); and providing predictable and guaranteed network performance with work-conserving sharing. To achieve these goals, it takes into account multiple types of resources to admit (allocate) applications in the datacenter without considering slots, so that tenants can request and receive the necessary amount of resources that their applications need to correctly execute and finish without delay and providers can avoid over-allocation.

Packer is designed with four aspects in mind: application abstraction, multi-resource allocation, network sharing and resource monitoring. First, Packer utilizes a novel abstraction for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). Unlike previous abstractions [2], [8]–[10], TI-MRA imposes no predefined structure for applications and allows the specification of requirements for multiple resources across time. Second, Packer employs a new allocation strategy that extends previous heuristics developed for multi-dimensional bin packing, in order to reduce multi-resource fragmentation. Third, Packer leverages Software-Defined Networking (SDN) and OpenFlow [11] to dynamically configure and enforce bandwidth guarantees for applications throughout the entire network. Fourth, Packer employs a monitoring mechanism to avoid resource wastage and to provide fast and up-to-date information upon unexpected events (e.g., if an application gets delayed due to a resource being congested).

Like Proteus [2] and the strategy in [7], Packer uses temporal resource demands to achieve maximum benefits. Consequently, it is better suited for applications that present a predefined behavior, repeatedly running the same type of tasks with similar input and data sets. This is common in iterative data processing (e.g., PageRank, hypertext-induced

topic search, recursive relational queries, social network analysis and network traffic analysis), where much of the data stays unchanged from iteration to iteration [2]. In this case, applications are profiled periodically or on each run. Furthermore, other applications can also take advantage of Packer by specifying only peak demands for multiple resources and, at runtime, employing its monitoring mechanism not to waste resources.

Overall, the major contributions of this paper are:

- A novel abstraction for applications, called Time-Interleaved Muti-Resource Abstraction (TI-MRA). In contrast to previous abstractions [2], [8]–[10], TI-MRA allows the specification of demands for multiple resources without a predefined structure for applications.
- A novel admission control algorithm that, by extending existing heuristics for multi-dimensional bin packing, minimizes resource fragmentation. The algorithm uses TI-MRA as input to coordinate requirements of applications in different resource dimensions across time.
- Packer, a scheme that combines TI-MRA and the novel admission control algorithm to provide predictable and guaranteed network performance with work-conserving sharing. Evaluation results show that it provides network performance guarantees, and improves datacenter utilization and provider revenue in comparison to related work, with the cost of taking more (yet acceptable) time to allocate applications.

II. PACKER

Packer implements a novel strategy for minimizing multi-resource fragmentation and for providing predictable and guaranteed network performance in large-scale cloud datacenters. Figure 1 shows an overview of Packer. The scheme is composed of three components: datacenter resource manager (DRM), OpenFlow-enabled switches and one local controller (i.e., a controller that integrates Open vSwitch and Algorithm 2) per server. They are discussed next.

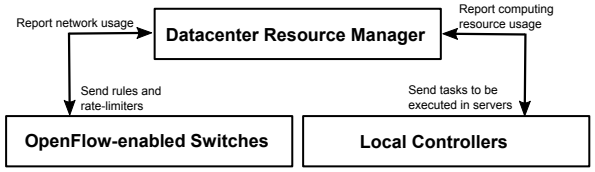


Fig. 1: Packer overview.

Datacenter resource manager (DRM). It is responsible for (i) allocating applications in the datacenter; and (ii) handling global events (e.g., bandwidth enforcement throughout the entire network for applications). More specifically, it receives an application request (in the form of a TI-MRA specification) and employs a novel resource allocation strategy (described in § II-B) to determine the set of resources to be used by the application. Then, it sends the application tasks to the servers that will execute them (as determined by the allocation strategy) and, via OpenFlow, configures switches (with rules and rate-limiters) to provide connectivity and network performance guarantees for the application. Furthermore, the DRM periodically receives up-to-date information regarding resource usage from local controllers at servers and from OpenFlow switches.

TABLE I: Notations adopted throughout the paper.

Symbol	Description
A	Set of application requests
G^a_{TI-MRA}	TI-MRA graph of application $a \in A$
V^a	Set of nodes of application $a \in A$ ($V^a = K^a \cup C^a$)
K^a	Set of tasks of application $a \in A$ ($K^a \subseteq V^a$)
C^a	Set of cloud services used by app $a \in A$ ($C^a \subset V^a$)
E^a	Set of edges (dependencies between nodes) of app $a \in A$
T^a	Discrete time instants of application $a \in A$ ($T^a \subseteq \mathcal{T}$)
w^a	Weight of application $a \in A$
N	Set of all infrastructure nodes ($N = S \cup \mathcal{J}$)
S	Set of servers in the datacenter infrastructure ($S \subseteq N$)
$\mathcal{J}$	Set of services available in the datacenter ($\mathcal{J} \subset N$)
$\mathcal{L}$	Set of links in the datacenter network
$\mathcal{T}$	Discrete time instants of the infrastructure
$\mathcal{P}$	Set of all paths available in the network
$\mathcal{P}(n_1, n_2)$	Set of paths from src node $n_1 \in N$ to dest node $n_2 \in N$
$w^{\mathtt{r}}$	Weight of $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}, \mathtt{BAND}\}$
$\delta(v, \mathtt{r}, t)$	Amount of resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ required by node $v \in V^a$ at time $t \in T^a$ for app $a \in A$
$\sigma(e = (u, v), t)$	Bandwidth for communication between nodes $u, v \in V^a \mid e = (u, v) \in E^a$ at time $t \in T^a$ for application $a \in A$
$\mathcal{M}_n(v)$	Node $n \in N$ that holds the node $v \in V^a$ of app $a \in A$
$\mathcal{M}_e(u, v)$	Infrastructure path ($\mathcal{P}(\mathcal{M}_n(u), \mathcal{M}_n(v))$) used for communication between nodes $u, v \in V^a$ of app $a \in A$
$\mathcal{A}(n)$	Tasks running at infrastructure node $n \in N$
$\mathcal{B}(l)$	Total capacity (bandwidth) of link $l \in \mathcal{L}$
$\mathcal{C}(l)$	Communications (edges in the TI-MRA) using link $l \in \mathcal{L}$
$\mathcal{D}(u)$	Application that task u belongs to
$\mathcal{E}(v)$	Nodes that node $v \in V^a \mid a \in A$ depends on
$\mathcal{Q}(n, \mathtt{r}, t)$	Amount of available resource $\mathtt{r}$ on $n \in N$ at time $t \in \mathcal{T}$
$\mathcal{R}(n, \mathtt{r})$	Capacity of infrastructure node $n \in N$ for resource type $\mathtt{r}$

OpenFlow switches. These devices are responsible for forwarding traffic according to the instructions received from the DRM. They receive rules and rate-limiters to correctly handle traffic and enforce bandwidth for applications. Moreover, they periodically report resource usage statistics to the DRM.

Local controllers (LCs) at servers. They are part of the resource monitoring mechanism utilized in Packer (described in § II-D). LCs are responsible for (i) monitoring multi-resource usage at servers and reporting it to the DRM; (ii) enforcing allocations; and (iii) reacting to local events (e.g., dealing with congested resources inside their respective server).

We detail Packer in the following manner. We first present the novel abstraction (called TI-MRA) used for applications in § II-A. Then, we utilize TI-MRA as input for the new allocation algorithm (§ II-B) and describe the strategy used for providing predictable and guaranteed network performance (§ II-C). Finally, we detail the resource monitoring mechanism in § II-D. The notations used throughout the paper are presented in Table I.

A. Time-Interleaved Multi-Resource Abstraction (TI-MRA)

Prior work has designed abstractions expressed as physical network models (i.e., the hose model) [2], [4], [8], two-level trees (hierarchical hose) [8], [10] or based on communication patterns (TAG) [9]. However, they focus on the network and neglect other resources. In particular, the hose model (used by most related work) does not accurately capture the network requirements of applications with complex traffic interactions [9].

An effective abstraction is expected to consider two purposes. The first is to allow tenants to specify their application requirements in a simple and accurate manner. The second is to allow providers to minimize over-allocation (i.e., allocating the correct amount of resources required by applications), which may increase the percentage of allocated applications and, consequently, may improve datacenter throughput.

Based on these purposes and the limitations of prior work, we propose a novel abstraction for applications, called Time-Interleaved Multi-Resource Abstraction (TI-MRA). TI-MRA allows the specification of not only network demands but also other types of resources. TI-MRA leverages tenants' knowledge of their applications to yield a flexible representation of the applications' resource consumption. It uses the same principle of (a) temporal bandwidth requirements in TIVC [2], but extends it to all kinds of resources; and (b) communication patterns in TAG [9]. Furthermore, it also takes into account dependencies other than among tasks (such as between tasks and cloud services), in order to optimize the use of resources. The intuition is that TI-MRA allows a flexible representation of application requirements rather than imposing a predefined abstraction (e.g., the hose model) for applications to map their requirements to.

TI-MRA extends the concept of time-varying graphs (TVGs) [12] to represent temporal demands of multiple resources. A TI-MRA graph of application $a \in A$ is represented as $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$, with the terms being defined as follows: $V^a = K^a \cup C^a$ is the set of application nodes, composed of tasks (K^a) and cloud services required by the application (C^a); E^a is the set of edges, representing the dependencies between nodes; T^a is the set of discrete time instants, from the time the first node of application a begins its computation to the time the last node finishes; $w^a \in [0, 1]$ indicates the weight of application a, so that residual resources (unallocated, or reserved resources for an application and not currently being used) can be proportionally shared among applications with more demands than their guarantees (work-conservation); and $\delta(v, \mathtt{r}, t) \in \mathbb{R}^+$ returns the demand of node $v \in V^a$ at time $t \in T^a$ for resource $\mathtt{r} \in \{\texttt{CPU}, \texttt{MEM}, \texttt{IO\_WRITE}, \texttt{IO\_READ}\}$. The last function, $\sigma(e = (u, v), t) \in \mathbb{R}^+$, denotes the bandwidth necessary for communication between nodes $u \in V^a$ and $v \in V^a \mid u \neq v$ at time $t \in T^a$, for $e = (u, v) \in E^a$. Note that we do not consider moving nodes and edges across time. This does not impact the generality of TI-MRA because when a node or edge has no demand for a given resource, the call for the respective function ($\delta(v, \mathtt{r}, t)$ or $\sigma(e = (u, v), t)$) returns zero.

An example of TI-MRA is shown in Figure 2. The figure depicts a simple application composed of five tasks and temporal resource requirements for CPU, memory, disk I/O write, disk I/O read and bandwidth. In this example, tasks T_1 and T_2 get their input data from storage service STS_1; T_3 depends on tasks T_1 and T_2 and on data sent from cloud service CS_1; T_4 reads data from storage service STS_2 to perform its computation; and T_5 depends on tasks T_3 and T_4 and stores the final result in STS_3. Moreover, note that edges (links representing the exchange of data) are unidirectional (different amounts of bandwidth for sending and receiving data). Having two links instead of a single bidirectional link avoids over-allocation and bandwidth wastage.

Producing TI-MRA models. TI-MRA can be used not only by tenants who have a deep understanding of their application demands, but also by users who do not know it in advance. The former can tune resource demands according to the application requirements, possibly reducing costs (avoiding over-allocation) without impact on performance. The latter, in turn, can specify only peak demands for resources (i.e., a constant temporal function). This would be similar to the hose model specification. Alternatively, the same strategy employed in CloudMirror [9] for generating TAG models could be used here: application templates for TI-MRA could be provided as a library for users through the extension of cloud orchestration systems like OpenStack Heat and AWS CloudFormation.

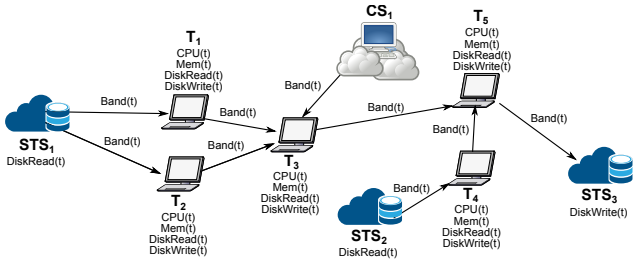


Fig. 2: TI-MRA of a simple application composed of five tasks (T_1 to T_5), where tasks read and write data from/to storage services (STS_1, STS_2 and STS_3) and use cloud service CS_1.

B. Allocation Strategy

The problem of allocating applications considering multiple types of resources in datacenters can be reduced to multi-dimensional bin packing (also called vector bin packing – VBP) [6], which is NP-Hard for every dimension d and APX-Hard for $d \geq 2$ [3]. Given balls (application tasks) and bins (servers) with sizes for each property (resource) considered, VBP assigns the balls to bins according to an optimization objective. In our case, the goal is to reduce fragmentation and over-provisioning and, consequently, improve the percentage of allocated applications and their tasks.

In case applications are constrained by a single resource (e.g., network), the problem becomes, in essence, a one dimensional bin packing [3]. However, cloud applications are typically constrained by multiple resources, including network [4] and CPU [13]. Moreover, the network is a distributed resource (composed of several links); therefore, the amount of resources consumed depends on bandwidth demands of tasks as well as their location in the infrastructure (the whole path used for communication must be taken into account), which increases the difficulty in efficiently optimizing the use of resources.

Problem definition. The process of multi-resource allocation is formally defined as follows. The TI-MRA specification of application $a \in A$ is given by $G^a_{TI-MRA} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. A node $v \in V^a$ can be either a task or a cloud service and is mapped to an infrastructure node $n \in N$. Each task $k \in K^a \mid K^a \subseteq V^a$ is assigned to an infrastructure server ($s \in S \mid S \subseteq N$) by mapping $\mathcal{M}_n : K^a \rightarrow S,\ \forall a \in A$ (Equation 1). Each cloud service $c \in C^a \mid C^a \subset V^a$ required by application $a \in A$, in turn, is part of the set of services ($\mathcal{J} \mid \mathcal{J} \subset N$) available in the platform (offered by either the provider or a tenant) and is assigned to a node $j \in \mathcal{J}$ that runs the requested service by mapping $\mathcal{M}_n : C^a \rightarrow \mathcal{J},\ \forall a \in A$ (Equation 2).

$$\mathcal{M}_n(k) \in S \mid k \in K^a \text{ or} \tag{1}$$

$$\mathcal{M}_n(c) \in \mathcal{J} \mid c \in C^a \tag{2}$$

Each dependency between nodes (i.e., edges in the TI-MRA graph, specified in set E^a) is mapped to a single path between the corresponding infrastructure nodes (servers for tasks and cloud services for services needed by the application). The assignment is defined by mapping $\mathcal{M}_e : E^a \rightarrow \mathcal{P},\ \forall a \in A$, where $\mathcal{P}$ denotes the set of all available paths in the network, such that for all $e = (u, v) \in E^a,\ \forall a \in A$:

$$\mathcal{M}_e(u, v) \in \mathcal{P}\left(\mathcal{M}_n(u), \mathcal{M}_n(v)\right) \tag{3}$$

The nodes in V^a run for at most T^a discrete time units to perform the computation required by application $a \in A$ and communicate among themselves using links $e \in E^a$. Each node $v \in V^a$ presents temporal demands for computing resource $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ ($\delta(v, \mathtt{r}, t)$), $\forall t \in T^a$. Furthermore, each edge $e = (u, v) \in E^a$ between communicating nodes u and v (such that $u \neq v$) presents temporal bandwidth demands: $\sigma(e = (u, v), t), \forall t \in T^a$.

A valid allocation is constrained by the amount of available resources. More specifically, a cloud service $j \in \mathcal{J}$ will only perform the task required by node $c \in C^a \mid a \in A$ if it has enough available resources to satisfy the demands. Similarly, a task can only be allocated in a server if the server has enough available capacity for each type of resource being considered. Given $\mathcal{A}(n)$ a function that returns all tasks running at infrastructure node $n \in N$, $\mathcal{R}(n, \mathtt{r})$ a function that returns the capacity of infrastructure node $n \in N$ for resource $\mathtt{r}$ and $\mathcal{T}$ the discrete time instants of the infrastructure, the constraint for computing resources is defined as follows:

$$\sum_{v \in \mathcal{A}(n)} \delta(v, \mathtt{r}, t) \leq \mathcal{R}(n, \mathtt{r}) \quad \forall n \in N,\ \forall \mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\},\ \forall t \in \mathcal{T} \tag{4}$$

The network is a special case, since it is a distributed resource. Specifically, bandwidth must be taken into account at the entire path used for each communication between application nodes and the amount of allocated bandwidth at a link l must not exceed the total capacity of l (Equation 5).

$$\sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e, t) \leq \mathcal{B}(l) \qquad \forall l \in \mathcal{L},\ \forall t \in \mathcal{T} \tag{5}$$

where $\mathcal{L}$ represents the set of links in the datacenter network, $\mathcal{C}(l)$ returns the set of communications (edges in TI-MRA graphs of applications) using link l and $\mathcal{B}(l)$ returns the total capacity (bandwidth) of link l.

Note that the above constraints are non-linear, in particular functions $\mathcal{A}$ and $\mathcal{C}$ (since they depend on the allocation of application nodes in infrastructure nodes). Efficient solvers are known only for some non-linear problems, such as the quadratic assignment problem. However, even when placement considerations are eliminated, the problem of VBP is APX-Hard [14] and re-solving it whenever new applications arrive worsens the process. Consequently, finding the optimal solution is expensive and requires a lot of time. Unfortunately, large-scale cloud datacenters require the allocation process to be performed as fast as possible, since they typically have high rate of application arrival and departure.

Algorithm. VBP has several proposed heuristics [3]. However, those heuristics cannot be used in datacenters without substantial modification, as they (i) assume that all input (i.e., applications) is known a priori, whereas we need to cope with online arrival of applications; (ii) consider "balls" of a fixed size, while applications have time-varying demands; and (iii) do not consider a distributed resource such as the network (with paths composed of multiple links). Therefore, we design a novel algorithm to efficiently allocate applications in cloud datacenters. The key principle is minimizing multi-resource fragmentation, thus improving the ratio of allocated applications and, consequently, maximizing datacenter utilization and provider revenue.

Algorithm 1 allocates applications as they arrive, in an online manner. It receives as input the datacenter infrastructure ($\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$) and the TI-MRA specification of an application $a \in A$ ($G^a_{\mathtt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$). First, it calculates available resources (CPU, memory, disk I/O write and disk I/O read) in servers (lines 1 – 2) and available bandwidth in links (lines 3 – 4). Since each node of application a may have a different duration, available resources in the infrastructure are calculated for T^a time units (the duration of a).

After that, nodes in V^a are sorted sequentially according to their initial execution time and the nodes they depend on (line 5). Based on the sorted list of nodes (`sNodes`), the algorithm gets one node v at a time (line 6), initializes as empty the list of infrastructure nodes with enough available resources to hold node v (line 7) and calculates, based on our novel metric shown in Equation 6, the score of v on infrastructure nodes (lines 8 – 11). The metric extends the ones in Panigrahy et al. [3] and works as follows. Equation 6 seeks to maximize the score achieved by both computing and network resources according to their current utilization level (calculated in Equations 7 and 8, respectively) in case there are enough available resources (i.e., node $n \in N$ and link $l \in \mathcal{L}$, which v would use for communication with the application nodes it depends on if it were allocated on n, have enough available resources at all times $t \in T^a$). Otherwise, it returns $-\infty$. For each computing resource $R = \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$ (Equation 7) and for bandwidth `BAND` (Equation 8), the metric subtracts the resource demand from the respective available resource, raises the resulting value by the power of 3, multiplies it by the amount of the respective available resource and multiplies it again by the weight associated to the resource ($w^{\mathtt{r}}$). In particular, $w^{\mathtt{r}}$ is dynamically defined as being inversely proportional to the current utilization level of $\mathtt{r}$ and is calculated as $w^{\mathtt{r}} = 1 - (\frac{\mathtt{util(r)}}{\sum_{s \in R \cup \{\mathtt{BAND}\}} \mathtt{util(s)}}), \forall \mathtt{r} \in R \cup \{\mathtt{BAND}\}$. That is, the higher the current utilization of $\mathtt{r}$, the lower its weight. This way, the metric prioritizes resources with lower utilization.

$$\mathtt{S}[n, v] = \begin{cases} \mathtt{S_C}[n, v] + \mathtt{S_B}[n, v] & \delta(v, \mathtt{r}, t) \leq \mathcal{Q}(n, \mathtt{r}, t) \text{ and } \sigma((u, v), t) \leq \mathcal{Q}(l, \mathtt{BAND}, t), \\ & \forall \mathtt{r} \in R, \forall u \in \mathcal{E}(v), \forall t \in T^a, \\ & \forall l \in \mathcal{M}_e(u, v); \\ -\infty & \text{otherwise.} \end{cases} \tag{6}$$

$$\mathtt{S_C}[n, v] = \sum_{t \in T^a} \sum_{\mathtt{r} \in R} w^{\mathtt{r}} * \left(\mathcal{Q}(n, \mathtt{r}, t) - \delta(v, \mathtt{r}, t)\right)^3 * \mathcal{Q}(n, \mathtt{r}, t) \tag{7}$$

$$\mathtt{S_B}[n, v] = \sum_{t \in T^a} \sum_{u \in \mathcal{E}(v)} \sum_{l \in \mathcal{M}_e(u,v)} w^{\mathtt{BAND}} * \left(\mathcal{Q}(l, \mathtt{BAND}, t) - \sigma((u, v), t)\right)^3 * \mathcal{Q}(l, \mathtt{BAND}, t) \tag{8}$$

Note that the metric described here uses normalized values (for resource requirements of applications as well as residual resources in the datacenter infrastructure) by the capacity of the node/link being considered.

The next step is the allocation of v and its communication

dependencies (edges with v as the destination node in the TI-MRA graph) in lines 12 – 17, according to Equations 1 – 5. Function `GetNodeWithBestScore` returns the infrastructure node with the best (maximum) score in the list `FeasibleNodes` for holding node v (line 12). In case no infrastructure node has enough available resources to hold v, the algorithm returns a failure code and application a is discarded (line 13). Otherwise, node v is allocated at infrastructure node n (line 15) and, since nodes that v depends on are already allocated (i.e., dependencies are allocated first, according to the sorted list of nodes), bandwidth for communication between v and its dependencies is also allocated (lines 16 – 17). When all nodes and edges in the TI-MRA graph of application a are successfully allocated, the algorithm returns a success code and finishes (line 18).

Algorithm 1: Multi-Resource Allocation Algorithm.

Input : Datacenter infrastructure $\langle N, \mathcal{L}, \mathcal{T}, \mathcal{P} \rangle$, Application a represented by $G^a_{\text{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$
Output: Success/Failure code

1 **foreach** *Infrastructure node* $n \in N$ **do**
2 $\quad \mathcal{Q}[n, \mathtt{r}, t] \leftarrow \mathcal{R}(n, \mathtt{r}) - \sum_{v \in \mathcal{A}(n)} \delta(v, \mathtt{r}, t),\ \forall \mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\},\ \forall t \in T^a \mid T^a \subseteq \mathcal{T}$;
3 **foreach** *Infrastructure link* $l \in \mathcal{L}$ **do**
4 $\quad \mathcal{Q}[l, \mathtt{BAND}, t] \leftarrow \mathcal{B}(l) - \sum_{e=(u,v) \in \mathcal{C}(l)} \sigma(e, t),\ \forall t \in T^a \mid T^a \subseteq \mathcal{T}$;
5 sNodes $\leftarrow$ SortNodes (V^a);
6 **foreach** $v \in$ sNodes **do**
7 $\quad$ FeasibleNodes $\leftarrow \emptyset$;
8 $\quad$ **foreach** $n \in N$ **do**
9 $\quad\quad$ S$[n, v] \leftarrow$ calculate score according to Equations 6, 7 and 8;
10 $\quad\quad$ **if** Score$[n, v] \neq -\infty$ **then**
11 $\quad\quad\quad$ FeasibleNodes $\leftarrow$ FeasibleNodes $\cup \{n\}$;
12 $\quad n \leftarrow$ GetNodeWithBestScore(FeasibleNodes);
13 $\quad$ **if** n *is null* **then return** failure code;
14 $\quad$ **else**
15 $\quad\quad \mathcal{M}_n(v) \leftarrow n$;
16 $\quad\quad$ **foreach** $u \in \mathcal{E}(v)$ **do**
17 $\quad\quad\quad \mathcal{M}_e(u, v) \leftarrow p \mid p \in \mathcal{P}\,(\mathcal{M}_n(u), \mathcal{M}_n(v))$;
18 **return** success code;

C. Network Sharing Strategy

The network sharing strategy has two objectives: (i) providing predictable and guaranteed network performance for applications, in order to avoid performance interference [4]; and (ii) achieving work-conserving sharing, so that applications have the possibility of using more bandwidth than their guarantees when needed and providers can achieve high network utilization.

To achieve these goals, we leverage the paradigm of SDN to dynamically configure the network, in order to enforce bandwidth guarantees and to provide work-conserving sharing. The strategy works as follows. According to the output of Algorithm 1 for application $a \in A$, the DRM performs two actions. First, it sends each task of a to the local controller of its selected server (as LCs manage and enforce resource allocation at servers). Second, it installs rules and rate-limiters in forwarding devices to guarantee connectivity and bandwidth for communication between these nodes.

In addition to ensuring a base level of guaranteed rate for applications, the strategy can proportionally share available bandwidth among applications with more demands than their guarantees. Towards this end, local controllers run an algorithm to periodically set the allowed rate for each allocated application node. Algorithm 2 aims at enabling smooth response to bursty traffic (since traffic in DCNs may be highly variable over short periods of time [15]). It receives as input the infrastructure node $n \in N$ that it belongs to, the current time $t \in \mathcal{T}$, current bandwidth demands of application nodes allocated at n (which are determined by monitoring socket buffers, similarly to Mahout [16]) and temporal bandwidth requirements of these nodes (specified in the request). First, the algorithm initializes as empty the list of application nodes with more bandwidth demands than the value specified in the request (line 1). Then, for each application node v allocated at n (line 2), the minimum rate between (i) the specified demand at time t ($\sigma(\sum(v, *), t)$, which represents the sum of bandwidth required by node v for communication with all nodes that depend on v at time t) and (ii) the current demand of v ($d[v]$) is assigned to `nRate` (line 3). If the current demand is higher than the specified demand, the node is added to the list of nodes with more demands than their guarantees (called `hungryNodes`, in line 4).

Algorithm 2: Work-conserving algorithm.

Input : Infrastructure node n, Time $t \in \mathcal{T}$, Current bandwidth demands of applications nodes d, Temporal bandwidth requirements of application nodes σ
Output: Rate `nRate` for each application node

1 hungryNodes $\leftarrow \emptyset$;
2 **foreach** $v \in \mathcal{A}(n)$ **do**
3 $\quad$ nRate[v] $\leftarrow \min(\sigma(\sum(v, *), t), d[v])$;
4 $\quad$ **if** $\sigma(\sum(v, *), t) < d[v]$ **then** hungryNodes $\leftarrow$ hungryNodes $\cup\ v$;
5 $\mathcal{Q}(n, \mathtt{BAND}, t) \leftarrow \mathcal{B}(link) - \sum_{v \in \mathcal{A}(n)}$ nRate[v], at time t;
6 **while** $\mathcal{Q}(n, \mathtt{BAND}, t) > 0$ *and* hungryNodes *not empty* **do**
7 $\quad$ **foreach** $v \in$ hungryNodes **do**
8 $\quad\quad$ value $\leftarrow \min\left(d[v] - \text{nRate[v]}, \left(\frac{w^{\mathcal{D}(v)}}{\sum_{u \in \text{hungryNodes}} w^{\mathcal{D}(u)}} \times \mathcal{Q}(n, \mathtt{BAND}, t)\right)\right)$;
9 $\quad\quad$ nRate[v] $\leftarrow$ nRate[v] + value;
10 $\quad\quad \mathcal{Q}(n, \mathtt{BAND}, t) \leftarrow \mathcal{Q}(n, \mathtt{BAND}, t)$ − value;
11 $\quad\quad$ **if** nRate[v] $== d[v]$ **then** hungryNodes $\leftarrow$ hungryNodes $\setminus \{v\}$;
12 **return** nRate;

Then, the algorithm calculates the residual bandwidth ($\mathcal{Q}(n, \mathtt{BAND}, t)$) of the wired link connecting server n to its top-of-rack (ToR) switch at time t (line 5). The residual bandwidth is calculated by subtracting from the link capacity the rate assigned to the application nodes (in line 3). The last step establishes the bandwidth rate for application nodes with more demands than their guarantees, if there is available bandwidth (lines 6 – 11). The rate of each node $v \in$ `hungryNodes` (in line 9) is determined by adding `nRate`[v] (initialized in line 3) and the minimum bandwidth between (i) the difference of the current demand ($d[v]$) and the rate (`nRate`[v]); and (ii) the proportional share of residual bandwidth the application node can receive according to its weight $w^{\mathcal{D}(v)}$ (calculated in line 8), where $\mathcal{D}(v)$ indicates the application that node v belongs to. The residual bandwidth is updated in line 10 and, in case the demands of node v were satisfied, it is removed from the list `hungryNodes` (line 11). Note that there is a "while" loop (lines 6 – 11) to guarantee that all residual bandwidth is used or all demands are satisfied. If this loop were not used, there could be occasions when there would be unsatisfied demands even though some bandwidth would be available.

In summary, if the demand of an application node exceeds its guaranteed rate (the rate specified in the request – σ), data can be sent and received at least at the guaranteed rate. Otherwise, if it does not, the unutilized bandwidth will be shared among co-resident application nodes whose traffic

demands exceed their guarantees (work-conservation).

Finally, note that SDN has scalability challenges on DCNs [11]: (i) elevated flow setup time, as forwarding devices ask the controller for appropriate rules when they receive the first packet of a new flow; and (ii) large flow tables in switches, since DCNs may have millions of flows per second [17] and, thus, the number of entries needed in TCAMs may be significantly higher than the amount of resources available in commodity switches. We adopt the strategy proposed in Marcon et al. [18] to address these challenges. The interested reader may refer to [18] for more details.

D. Resource Monitoring Mechanism

Packer is designed with scalability and high multi-resource utilization (i.e., minimizing fragmentation of multiple resources) in mind. This implies that the resource monitoring mechanism (i) should not incur significant overhead (especially to scarce resources such as the network [2]); and (ii) needs to be able to acquire real-time information about resource usage, so that idle resources can be allocated to applications that need them. Moreover, the mechanism is expected to provide fast and up-to-date information upon unexpected events (e.g., in case an application gets delayed due to a resource being congested).

We designed a two-level strategy for resource monitoring, composed of (i) the DRM and (ii) local controllers at servers and OpenFlow switches. First, a local controller runs at each server and coordinates the allocation of the server's resources to application tasks. LCs have two objectives, described as follows. The first objective is to observe aggregate resource usage and periodically report it to the DRM (so that the DRM gets updated information about the infrastructure utilization). The second objective is related to handling local events: since LCs have no interconnection among themselves (in order to reduce management traffic in the network) and no knowledge of infrastructure-wide state, they are allowed to handle only local events (e.g., dealing with local congested resources and enforcing allocations to tasks). This is important for relieving the load on the DRM and for reducing the amount of bandwidth used for communication between LCs and the DRM.

Second, the DRM maintains infrastructure-wide state, as it periodically receives resource usage statistics from local controllers at servers and from switches (via the OpenFlow protocol). With the information received from servers and switches, it reacts to global events such as the allocation of applications and bandwidth enforcement throughout the entire network for applications.

III. Evaluation

In this section, we focus on showing that Packer (i) minimizes multi-resource fragmentation; (ii) improves provider revenue; (iii) incurs acceptable overhead; (iv) provides predictable and guaranteed network performance with work-conserving sharing; and (v) outperforms existing state-of-the-art schemes (Tetris [6] and slot-based allocation [19], [20]).

A. Setup

Environment. We have implemented a simulator that models computing and network resources of a multi-tenant datacenter. For computing resources, we follow Tetris [6] and use a similar server configuration: 16 CPU cores, 32 GB of memory, 4 disks operating at 50 MBps each for read and write operations and a 1 Gbps NIC. For the slot-based scheme, we follow related work [8] and divide each server into four equal slots for VMs. The network, in turn, is defined as a tree-like topology, similar to current DCNs and related work [4]. It is composed of a three-tier topology with 1,200 servers at level 0. Every 40 machines form a rack, and every 10 ToRs are connected to an aggregation switch. Finally, all aggregation switches are connected to a core switch. Unless otherwise specified, the capacity of each link is defined as follows: 1 Gbps for server-ToR links, 10 Gbps for ToR-aggregation links and 100 Gbps for aggregation-core links.

Workload. We built a workload suite composed of incoming application requests (to be allocated in the datacenter) arriving over time. We consider a heterogeneous set of applications, including MapReduce and Web Services. As defined in § II-A, each application a is represented by a TI-MRA graph $G^a_{\texttt{TI-MRA}} = \langle V^a, E^a, T^a, w^a, \delta, \sigma \rangle$. Given the lack of publicly available traces for DCNs, the workload was generated in line with related work [2], [6], [17], [21], [22]. First, like Tetris [6], computing resources of tasks were picked uniformly at random between 0 and the maximum value of a slot. Note that we limit the demand for computing resources of each task from each application to the maximum size of a slot in order to provide a fair and accurate comparison (otherwise, since we use the same workload for all schemes, some tasks would never be allocated with the slot-based approach). Second, bandwidth demands were generated based on the measurements from Benson et al. [17] and Kandula et al. [22]. Finally, the weight w^a of each application a is uniformly distributed in the interval $[0, 1]$.

B. Results

We compare Packer with Tetris [6] and the slot-based allocation [19], [20]. For all experiments comparing different strategies, we plot the percentage improvement (or reduction) between Packer and the related work being compared as $\frac{\texttt{Packer}-\texttt{related\_work}}{\texttt{Packer}} * 100\%$. Hence, positive values mean Packer has achieved a higher value than the approach being compared, while negative values mean Packer has achieved a lower value. In general, higher values are better, with the sole exception being the overhead of the allocation algorithm (Figure 6).

Increased acceptance ratio. Figure 3 shows the proportion of application tasks that were allocated between Packer and Tetris and Packer and slot-based according to the time. *Higher values are better*, as they mean that Packer allocates more tasks than the respective proposal being compared. At first, the gains of Packer in comparison to the other proposals have high variability because there are ample resources and, therefore, most incoming applications are allocated. As time passes and the cloud-load increases (less available resources), the gains tend to stabilize (around time 1,000), because new applications are allocated only when already allocated applications conclude their execution and are deallocated (which releases resources). In general, we observe that Packer consistently outperforms Tetris (≈30%) and slot-based allocation (≈67%). Although the amount of available resources in the infrastructure is the same, the allocation ratio differs for each approach. This happens because each scheme uses a different allocation strategy. More specifically, Tetris seeks to minimize computing resource fragmentation, while only penalizing the bandwidth used. This may not result in good choices for allocation because of network fragmentation (as the network

is an important bottleneck in datacenters [2]). The slot-based allocation, in turn, is constrained by the static number and size of slots in the servers, which limit the feasible choices for allocating tasks to servers. In contrast to both proposals, Packer employs our novel algorithm described in § II-B and better explores the trade-off between using local computing resources (CPU, memory and disk I/O) and remote distributed resources (the network).

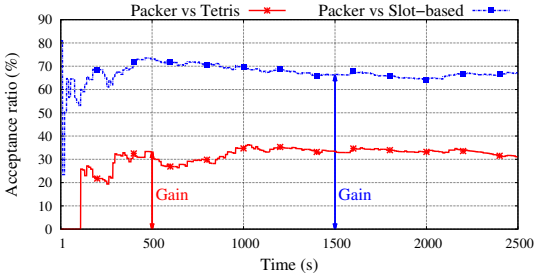


Fig. 3: Acceptance ratio of application tasks.

Maximized resource utilization. Figure 4 depicts the percentage difference between Packer and Tetris and Packer and slot-based allocation for the utilization of different types of resources. Positive values indicate that Packer achieves better utilization than the respective proposal being compared (*the higher the value, the better*), while negative values denote that the proposal being compared achieves better results than Packer. Figures 4(a), 4(b), 4(c) and 4(d) show results for CPU, memory, disk I/O write and disk I/O read, respectively. We see that, during most of the time, Packer allows better use of available resources, improving utilization by a significant percentage. Nonetheless, Packer has lower utilization of some types of resources at some periods of time (negative values in the plots, e.g., at time 500s). This happens because, with different allocation strategies, different applications are accepted and rejected in each scheme. Therefore, despite allocating significantly more application tasks than Tetris and slot-based (Figure 3), there are some small periods of time when the tasks allocated in Packer consume less resources.

Figure 4(e), in turn, shows the percentage difference of bandwidth utilization. We verify that Packer can maintain significantly higher utilization of the network than Tetris (≈76% more) and slot-based (≈87% more). Moreover, during the experiments, Packer never achieved lower network utilization than the proposals being compared. In general, Figures 3 and 4 show that Packer accepts more applications and, consequently, maximizes utilization, which indicates that fragmentation is minimized.

Increased provider revenue. We follow related work [2], [8] and adopt a simple pricing model to quantify provider revenue for Packer, Tetris and slot-based allocation, which effectively charges both computation and networking. A tenant running application a pays: $\sum_{t \in T^a} \sum_{v \in V^a} (\sum_{\mathtt{r}} \delta(v, \mathtt{r}, t) * k_v + \sum_{u \in \mathcal{E}(v)} \sigma((u,v), t) * k_b)$, where $\mathtt{r} \in \{\mathtt{CPU}, \mathtt{MEM}, \mathtt{IO\_WRITE}, \mathtt{IO\_READ}\}$, k_v is the unit-time computing resource cost and k_b is the unit-volume bandwidth cost. Figure 5 depicts the revenue of Packer in comparison to Tetris and slot-based allocation (error bars show 95% confidence interval). *Higher values are better*, as they mean that Packer provides more revenue than the respective proposal being compared. We see that, by improving the allocation ratio of application tasks (Figure 3) and resource utilization (Figure 4), Packer can significantly increase provider revenue (≈29% and ≈60% in comparison to Tetris and slot-based, respectively).

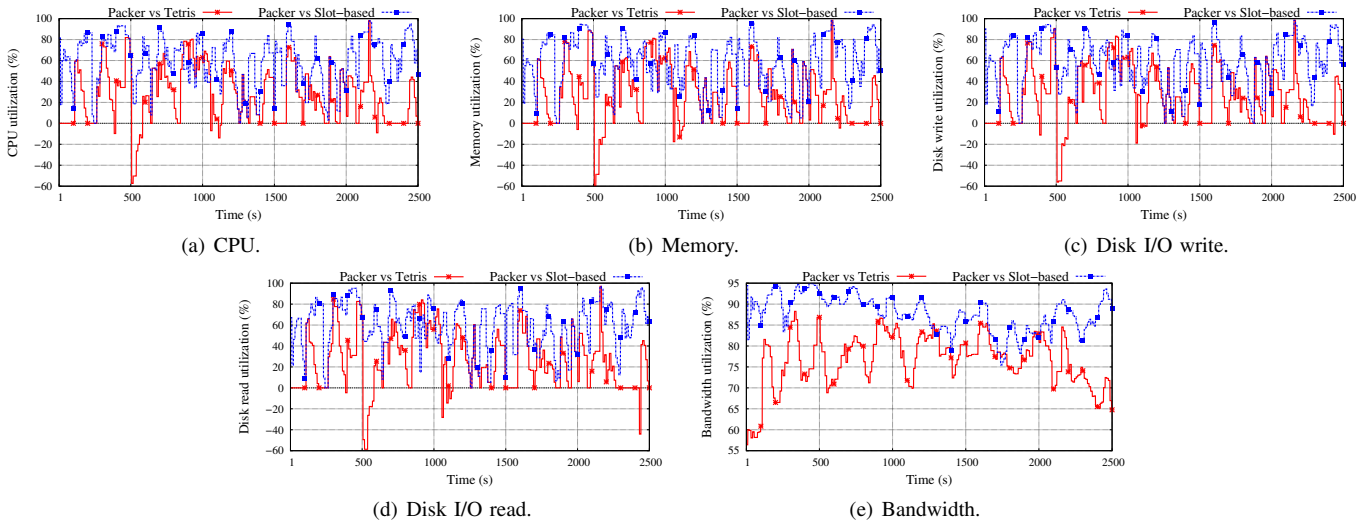
Acceptable overhead. Figure 6 quantifies the overhead introduced by Packer in comparison to Tetris and slot-based (error bars show 95% confidence interval). The overhead is given by the mean time taken to allocate an incoming application in the infrastructure. Here, positive values indicate that Packer takes more time to allocate applications than the respective proposal being compared, while negative values would indicate that Packer takes less time (i.e., *lower values are better*). We see that Packer takes more time to allocate applications than the other two proposals (≈53% more time than Tetris and ≈81% more than slot-based). This is justified by three factors (i) the complexity of the allocation metric (§ II-B); (ii) the fact that Packer considers the whole network (as opposed to Tetris that only penalizes network use); and (iii) the fact that Packer verifies each computing resource (CPU, memory and disk I/O) according to the applications' requirements (as opposed to slot-based that statically divides computing resources into slots). Nonetheless, while the percentage is high, the median time taken to allocate applications (observed in our experiments) is small for all three proposals: ≈15.4s in Packer, ≈3.5s in Tetris and ≈0.3s in slot-based allocation. Thus, considering the benefits provided by Packer (shown in Figures 3, 4 and 5), it is acceptable to take some additional seconds to allocate applications.

Now, we turn our focus to the challenge of performance interference. In particular, we show that Packer provides (i) minimum bandwidth guarantees for applications; and (ii) work-conserving sharing, achieving both predictability for tenants and high utilization for providers. To show the results in a clear way, here we consider the requested temporal bandwidth guarantees of application tasks (σ) as a constant function (while the actual requirements vary over time).

Minimum bandwidth guarantees for applications. Packer adopts the following definition of minimum bandwidth guarantees: the task rate should be (a) at least the guaranteed rate if the demands are equal or higher than the guarantees; or (b) equal to the demands if they are lower than the guarantees. To illustrate it, we show, in Figure 7, a task allocated on a given server during a predefined time period of an experiment. We see that the task may not get the desired rate to satisfy all of its demands instantaneously (when its demands exceed its guarantees) because (i) the link capacity is limited; and (ii) available bandwidth is proportionally shared among tasks. In summary, we verify that Packer provides minimum bandwidth guarantees for tasks, since the actual rate is always equal or higher than the minimum between the demands and the guarantees. Therefore, applications have minimum bandwidth guarantees and, thus, can achieve predictable network performance.

Work-conserving sharing. Work-conservation is the ability to use more bandwidth if the task has higher demands than its guarantees and there is available bandwidth in the network. In other words, bandwidth which is not allocated, or allocated but not currently used, should be proportionally shared among tasks with more demands than their guarantees (according to the weights of each application – w^a, using Algorithm 2). Figure 8 shows the aggregate bandwidth[1] on the server holding

[1]Note that Packer considers the temporal bandwidth guarantees requested (σ) when allocating tasks. Therefore, although the sum of the actual demands of all tasks allocated on a given server may exceed the server link capacity, the sum of bandwidth guarantees of these tasks will not exceed the link capacity.



(a) CPU. (b) Memory. (c) Disk I/O write.

(d) Disk I/O read. (e) Bandwidth.

Fig. 4: Resource utilization (positive values in the y-axis indicate that Packer achieves better utilization than the respective proposal being compared, while negative values denote that the proposal being compared achieves better utilization than Packer).

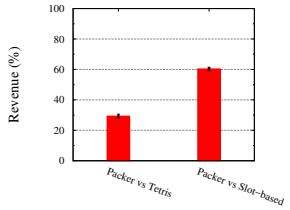


Fig. 5: Revenue.

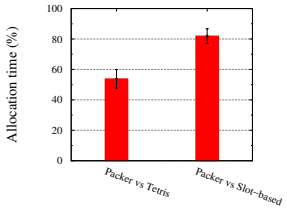


Fig. 6: Allocation time.

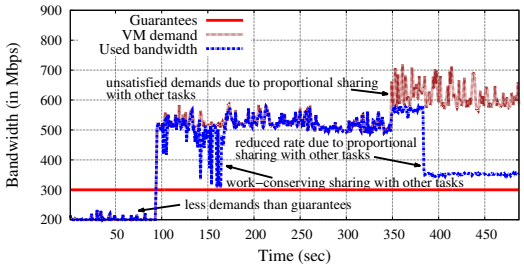


Fig. 7: Bandwidth allocation for a task on a given server.

the task in Figure 7. In these two figures, we verify that Packer provides work-conserving sharing in the network, as tasks can receive more bandwidth (if their demands are higher than their guarantees) when there is spare bandwidth. Thus, providers can achieve high utilization.

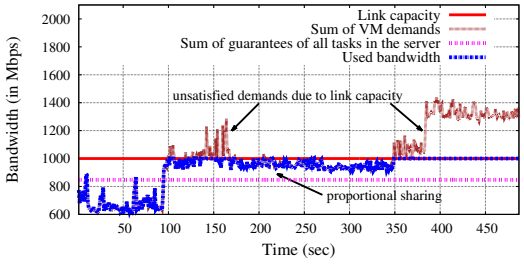


Fig. 8: Work-conserving sharing on a given server.

IV. Discussion

We discuss here some questions that have arisen during the design of Packer.

TI-MRA pros and cons. This abstraction considers temporal demands of resources. There are advantages and drawbacks of adopting this approach. The main advantage is to minimize multi-resource underutilization, which significantly improves datacenter utilization (i.e., reduces wastage). The main drawback is the need for fine-grained specification of applications, which may be a burden on some tenants (the ones with less knowledge of their applications). Hence, TI-MRA also allows the specification of only peak demands for multiple resources, minimizing the burden of application specification. While this may reduce infrastructure utilization, it does not impact provider revenue, as tenants are allocating such resources and paying for them.

Profiling application demands for specifying TI-MRA. Datacenter application demands are often known [23] or can be obtained from tenants [4], [8]. Alternatively, we employ the techniques described by Grandl et al. [6], Chen and Shen [7] and Lee et al. [9]. First, according to Chen and Shen [7], the same task (i.e., the same program with the same options) running on different servers tends to have similar resource utilization patterns. In this context, recurring applications are common in datacenters [24]; for instance, analytic applications repeat hourly or daily to perform the same computation on new data [6]. Therefore, Packer can use statistics measured in prior runs of the same application. Second, according to Lee et al. [9], orchestration systems like OpenStack Heat and AWS CloudFormation could be used to generate abstract models. They use templates (provided as a library for tenants) that explicitly describe the structure of applications and their resource demands. In this sense, these systems could be extended with temporal multi-resource requirements to generate TI-MRAs. Third, Packer can use the pattern detection algorithm for resource demands developed by Chen and Shen [7]. The algorithm utilizes logs of resource usage recorded by the cloud datacenter from previous runs of the same application and, thereby, can estimate utilization patterns for the requested application. Fourth, in case none of the previous methods can

be used, we follow Grandl et al. [6] and over-estimate resource demands (by considering a constant temporal function). Note that over-estimation is better than under-estimation, as the former does not slow down applications. Furthermore, Packer's resource monitoring mechanism verifies idle resources and reports them to the DRM, so that they can be allocated to applications.

Employing existing abstractions in the literature for Packer. Packer could use existing abstractions for application specification, with the constraint that those abstractions take into account multiple types of resources. Adapting other abstractions for Packer could be performed with the development of a module that reads the specification and converts it to a TI-MRA, so that the use of other abstractions would be seamless to Packer's allocation process.

V. Related Work

Proposals related to Packer are divided in two categories, as follows.

Multi-resource allocation. On one hand, schemes such as [19], [20] allocate computing resources based on slots. However, this leads to wastage and fragmentation, as the amount of resources in each slot is statically defined. On the other hand, Tetris [6], Dominant Resource Fairness (DRF) [1] and the spatial/temporal strategy [7] propose to dynamically adjust the allocation according to resource demands. Nonetheless, they present the following drawbacks: Tetris may result in starvation (depending on the workload) and may not provide guarantees in oversubscribed networks; DRF may result in fragmentation [6], as it focuses solely on fairness; and the strategy in [7] considers only temporal demands of CPU and memory, but neglects the network. Packer, in contrast, minimizes fragmentation (unlike slot-based schemes and DRF) and provides bandwidth guarantees even in oversubscribed networks (unlike Tetris and the strategy in [7]).

Network performance in DCNs. There is an extensive body of literature that addresses network performance in DCNs. We focus on the most important proposals related to Packer. Oktopus [8], CloudMirror [9] and Silo [4] provide network guarantees for applications. However, they focus only on network resources and may result in underutilization, as they statically reserve resources for applications based on their peak bandwidth demands. Proteus, in turn, allocates applications according to their temporal network demands. Despite reducing network underutilization, it neither considers other types of resources nor provides work-conserving sharing among applications (i.e., it uses rigid network models for each allocated application). Unlike Oktopus, CloudMirror, Silo and Proteus, Packer considers multiple types of resources and provides work-conserving sharing.

VI. Final Remarks

In this paper, we introduced Packer, a scheme that addresses the challenges of multi-resource allocation and performance interference in the network. It employs a novel abstraction called Time-Interleaved Multi-Resource Abstraction (TI-MRA) and a new algorithm for allocating multiple types of resources with reduced fragmentation. Furthermore, Packer uses (a) SDN to dynamically configure and manage the network according to available resources and requirements of applications; and (b) a monitoring mechanism to avoid wastage and congested resources. Evaluation results show that (i) acceptance ratio of applications is increased; (ii) datacenter utilization is maximized (i.e., fragmentation is minimized); (iii) provider revenue is augmented; and (iv) applications achieve predictable and guaranteed network performance with work-conserving sharing.

Acknowledgements

This work has been supported by MCTI/CNPq/Universal (Project Phoenix, 460322/2014-1) and Microsoft Azure for Research grant award.

References

[1] A. Ghodsi et al., "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types," in *USENIX NSDI*, 2011.

[2] D. Xie et al., "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *ACM SIGCOMM*, 2012.

[3] R. Panigrahy et al., "Heuristics for Vector Bin Packing," Tech. Rep., 2011, available at : http://goo.gl/9BDXIL.

[4] K. Jang et al., "Silo: Predictable Message Latency in the Cloud," in *ACM SIGCOMM*, 2015.

[5] M. Chowdhury et al., "HUG: Multi-Resource Fairness for Correlated and Elastic Demands," in *USENIX NSDI*, 2016.

[6] R. Grandl et al., "Multi-resource Packing for Cluster Schedulers," in *ACM SIGCOMM*, 2014.

[7] L. Chen and H. Shen, "Consolidating Complementary VMs with Spatial/Temporal-awareness in Cloud Datacenters," in *IEEE INFOCOM*, 2014.

[8] H. Ballani et al., "Towards predictable datacenter networks," in *ACM SIGCOMM*, 2011.

[9] J. Lee et al., "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, 2014.

[10] H. Ballani et al., "Chatty tenants and the cloud network sharing problem," in *USENIX NSDI*, 2013.

[11] Y. Jarraya et al., "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. PP, no. 99, pp. 1–29, 2014.

[12] K. Wehmuth et al., "A unifying model for representing time-varying graphs," 2015.

[13] K. Ousterhout et al., "Making Sense of Performance in Data Analytics Frameworks," in *USENIX NSDI*, 2015.

[14] G. J. Woeginger, "There is no asymptotic PTAS for two-dimensional vector packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

[15] D. Abts and B. Felderman, "A guided tour of data-center networking," *Commun. ACM*, vol. 55, no. 6, pp. 44–51, Jun. 2012.

[16] A. R. Curtis et al., "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*, 2011.

[17] T. Benson et al., "Network traffic characteristics of data centers in the wild," in *ACM IMC*, 2010.

[18] D. S. Marcon and M. P. Barcellos, "Predictor: providing fine-grained management and predictability in multi-tenant datacenter networks," in *IFIP/IEEE IM*, 2015.

[19] "Hadoop Scheduler," The Apache Software Foundation, 2014, available at : http://bit.ly/1tGpbDN, http://bit.ly/1tGpbDN.

[20] "Hadoop YARN Project," The Apache Software Foundation, 2015, available at : http://bit.ly/1iS8xvP.

[21] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *USENIX NSDI*, 2011.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM IMC*. New York, NY, USA: ACM, 2009.

[23] M. P. Grosvenor et al., "Queues Don't Matter When You Can JUMP Them!" in *USENIX NSDI*, 2015.

[24] S. Agarwal et al., "Re-optimizing Data-parallel Computing," in *USENIX NSDI*, 2012.