Colocation, Colocation, Colocation*: Optimizing Placement in the Hybrid Cloud

Srinivas Aiyar
Nutanix, Inc., San Jose, CA, USA
sriniva.aiyar, karan.gupta@nutanix.com

Karan Gupta
Nutanix, Inc., San Jose, CA, USA
karan.gupta@nutanix.com

Rajmohan Rajaraman
Northeastern University, Boston, MA, USA
rra@ccs.neu.edu

Bochao Shen
Northeastern University, Boston, MA, USA
ordinary@ccs.neu.edu

Zhifeng Sun
Northeastern University, Boston, MA, USA
austin@ccs.neu.edu

Ravi Sundaram
Northeastern University, Boston, MA, USA
koods@ccs.neu.edu

Abstract

Today's enterprise customer has to decide how to distribute her services among multiple clouds - between on-premise private clouds and public clouds - so as to optimize different objectives, e.g., minimizing bottleneck resource usage, maintenance downtime, bandwidth usage or privacy leakage. These use cases motivate a general formulation, the *uncapacitated*[1] multidimensional load assignment problem - VITA(F) (Vectors-In-Total Assignment): the input consists of n, d-dimensional load vectors $\bar{V} = \{\bar{V}_i | 1 \leq i \leq n\}$, m cloud buckets $B = \{B_j | 1 \leq j \leq m\}$ with associated weights w_j and assignment constraints represented by a bipartite graph $G = (\bar{V} \cup B, E \subseteq \bar{V} \times B)$ restricting load $\bar{V}_i$ to be assigned only to buckets B_j with which it shares an edge[2]. F can be any operator mapping a vector to a scalar, e.g., max, min, etc. The objective is to partition the vectors among the buckets, respecting assignment constraints, so as to achieve

$$\min[\sum_j w_j * F(\sum_{\bar{V}_i \in B_j} \bar{V}_i)]$$

We characterize the complexity of VITA(min), VITA(max), VITA(max − min) and VITA(2^{nd} max) by providing hardness results and approximation algorithms, *LP-Approx* involving clever rounding of carefully crafted linear programs. Employing real-world traces from Nutanix, a leading hybrid cloud provider, we perform a comprehensive comparative evaluation versus three natural heuristics - *Conservative*, *Greedy* and *Local-Search*. Our main finding is that on real-world workloads too, *LP-Approx* outperforms the heuristics, in terms of quality, in all but one case.

2012 ACM Subject Classification Modeling methodologies

* Play on the real estate mantra: Location, location, location
[1] A defining feature of clouds is their *elasticity* or ability to scale with load
[2] In a slight abuse of notation, we let B_j also denote the subset of vectors assigned to bucket B_j

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases Approximation algorithm, Vector packing, LP rounding

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

The launch of EC2 in 2006 by AWS [1] heralded the explosive growth in cloud computing. Cloud computing is an umbrella term for computing as an utility. It enables 24x7 Internet-based access to shared pools of configurable system resources and real-time provision-able higher-level services. Public clouds enable organizations to focus on their core businesses instead of spending time and money on IT infrastructure and maintenance. One of the major benefits of clouds is that they are *elastic*[3] (which we model in this paper as uncapacitated). This allows enterprises to get their applications[4] up and running quicker, and rapidly adjust resources to meet fluctuating and unpredictable business demand.

Today, in addition to AWS, Microsoft's Azure [5] and the Google Cloud [3] are the other major public cloud platforms. But the advent of multiple clouds means that enterprises are faced with several new questions, of which the following are some examples: How much of their load should they keep on-premise and how much should they colocate (or place) in public clouds? How should they mix and match the various options to save money without sacrificing customer satisfaction? A number of enterprise software companies such as HPE [4] and startups such as Turbonomic [7], Datadog [2] and RightScale [6] are beginning to provide software and sevice solutions to these problems.

At the same time this is also a fertile area for new problems with the potential for clever theoretical solutions to have practical impact. In this paper we provide a framework - VITA : Vectors-In-Total Assignment - that captures a variety of interesting problems in the area of hybrid clouds with interesting theoretical challenges. In the subsection that follows we list a few typical use cases captured by the VITA framework.

1.1 Motivation and Model

Scenario 1. Minimizing peak pricing: Consider an enterprise customer that has a choice of several different cloud providers at which to host their VMs (virtual machines). The requirements of each VM can be characterized along several different resource dimensions such as compute (CPU), network (latency, bandwidth), storage(memory, disk) and energy. When different virtual machines are placed in the same elastic resource pool (cloud), their load across each dimension is accrued additively (though, of course the different dimensions can be scaled suitably to make them comparable). A typical pricing contract will charge based on the most bottle-necked dimension since peak provisioning is the biggest and most expensive challenge for the resource provider. And different providers may have different rates based on differing infrastructure and their cost for installation and maintenance. The natural question then arises - what is the optimal way for the enterprise customer to distribute the load amongst the different cloud providers so as to minimize total cost?

Scenario 2. Minimizing maintenance downtime: Hosts and services, (and occasionally even data centers) need to be powered down every so often for maintenance purposes, e.g. upgrading the software version (or installing a new HVAC system in a data center).

[3] Elastic usually means that clouds can be considered to have infinite capacity for the operating range of their customers. In this paper we ignore fine-grained time-based definitions such as in [21]

[4] In the scope of this paper *application* refers to a collection of VMs and containers working in concert

Given this reality, how should the application (collection of virtual machines and/or containers collectively performing a task or service), be allocated to the different hosts so as to minimize the aggregate disruption? This scenario also applies to industrial machines where different factories (or floors of a factory) need to be shut down for periodical maintenance work.

Scenario 3. Preserving privacy: Consider a set of end-users each with its own (hourly) traffic profile accessing an application. We wish to partition the application components across a set of clouds such that by observing the (hourly volume of) traffic flow of any single cloud it is not possible to infer which components are colocated there. This leads to the following question - how should we distribute load across clouds in order to minimize the maximum hourly variation in aggregate traffic? As an analogy, the situation here is similar to the problem of grouping households such that the variation of energy usage of a group is minimized making it difficult for thieves to infer who has gone on vacation.

Scenario 4. Burstable billing: Most Tier 1 Internet Service Providers (ISPs) use burstable billing for measuring bandwidth based on peak usage. The typical practice is to measure bandwidth usage at regular intervals (say 5 minutes) and then use the 95th percentile as a measure of the sustained flow for which to charge. The 95th percentile method more closely reflects the needed capacity of the link in question than tracking by other methods such as mean or maximum rate. The bytes that make up the packets themselves do not actually cost money, but the link and the infrastructure on either end of the link cost money to set up and support. The top 5% of samples are ignored as they are considered to represent transient bursts. Burstable billing is commonly used in peering arrangements between corporate networks. What is the optimal way to distribute load among a collection of clouds, public and private, so as to minimize the aggregate bandwidth bill?

The above scenarios constitute representative situations captured by the *uncapacitated* multidimensional load assignment problem framework - VITA. A host of related problems from a variety of contexts can be abstracted and modeled as VITA(F): the input consists of n, d-dimensional load vectors $\bar{V} = \{\bar{V}_i | 1 \leq i \leq n\}$ and m cloud buckets $B = \{B_j | 1 \leq j \leq m\}$ with associated weights w_j and assignment constraints represented by a bipartite graph $G = (\bar{V} \cup B, E \subseteq \bar{V} \times B)$ that restricts load $\bar{V}_i$ to be assigned only to those buckets B_j with which it shares an edge. Here, F can be any operator mapping a vector to a scalar, such as projection operators, max, min, etc. Then the goal is to partition the vectors among the buckets, respecting the assignment constraints, so as to minimize

$$\sum_j w_j * F(\sum_{\bar{V}_i \in B_j} \bar{V}_i)$$

where, in a slight abuse of notation, we let B_j also denote the subset of vectors assigned to bucket B_j. VITA stands for Vectors-In-Total Assignment capturing the problem essence - vectors assigned to each bucket are totaled. Unless otherwise specified we use i to index the load vectors, j to index the cloud buckets and k to index the dimension. We let $\bar{V}_i(k)$ denote the value in the k'th position of the vector $\bar{V}_i$.

We now explain how VITA(F) captures the aforementioned scenarios. In general, dimensions will either represent categorical entities such as resources (e.g., CPU, I/O, storage, etc.,) or time periods (e.g., hours of the day or 5-minute intervals, etc.,). We gently remind the reader to note that in each of the scenarios the elasticity of the clouds is a critical ingredient so that contention between vectors is not the issue. The set of scenarios we present are but a small sample to showcase the versatility and wide applicability of the VITA framework.

Scenario 1 is captured by having a vector for each VM, with each dimension representing its resource requirement[5]; constraints representing placement or affinity requirements [22], weights w_j representing the rates at different cloud providers. Then minimizing the sum of prices paid for peak resource usage at each cloud is just the problem VITA(max).

In Scenario 2 each dimension represents the resource (say, CPU utilization) consumed by the application in a given time period, e.g. the vector for an application could have 24 dimensions one for each hour in the day. Once the application is assigned to a data center (or cloud or cluster) it is clear that disruption is minimized if the maintenance downtime is scheduled in that hour where total resource utilization is minimum. Then minimizing the aggregate disruption is captured by the problem VITA(min).

The dimensions in Scenario 3 are the hours of the day and the resource in question is the traffic. To prevent leakage of privacy through traffic analysis the goal is to distribute the application components across clouds so that the range between the peak and trough of traffic minimized. This problem is exactly represented as VITA($\max - \min$).

In Scenario 4, we have vectors for each application with 20 dimensions one for each 5th percentile [29, 28] or ventile of the billing period[6]. Then minimizing the aggregate bandwidth bill under the burstable, or 95th percentile, billing method is VITA(2^{nd} max).

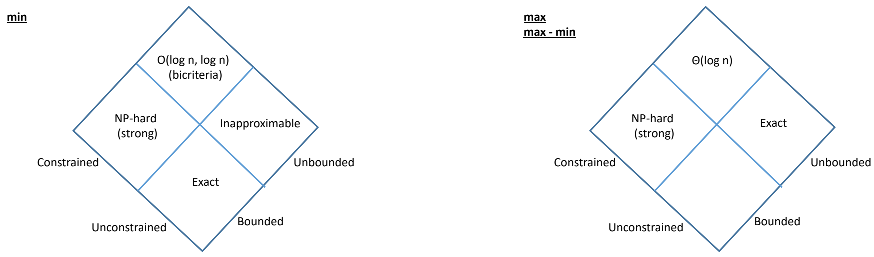
1.2 Our results

All the problems we consider are in NP [19]. For VITA(min) and VITA(max) we present our results as a lattice - see Figs 1a and 1b. For any given F, VITA(F) can be partitioned into a lattice of 4 different problem spaces based on the following 2 criteria: 1. constraints, and 2. dimensionality. The 4 different problem spaces arise from the Cartesian product: {unconstrained, constrained} X {bounded, unbounded}. *Unconstrained* refers to the situation where there is no bipartite graph representing constraints, i.e. any load vector may be placed in any bucket. And, *Bounded* refers to the situation where each load vector has a fixed dimension (independent of n). It should be clear that the simplest of the 4 spaces is unconstrained, bounded VITA(F) and the most general is the constrained, unbounded version of VITA(F). We present our results, algorithms and hardness, for the different F, in the form of a lattice. In each of the figures, the algorithmic results are displayed only at the highest possible node in the lattice, since it automatically applies to all nodes in the downward-closure; similarly, hardness results are presented at the lowest possible node since they apply to all nodes in the upward-closure. Further, our hardness results use only uniform weights whereas our algorithmic results work for general weights.

Our theory results are as follows:

- VITA(F) for F linear. We show that when F is linear then the problem is solvable exactly in polynomial-time. In particular VITA(avg) is in P.
- VITA(min). Our results are summarized in Fig. 1a. We show that VITA(min) is inapproximable when the dimensions are unbounded, i.e. it cannot be approximated to any finite factor. Since it is inapproximable we counter-balance this result by providing an $O(\log n, \log n)$-bicriteria approximation algorithm [25]. Our bicriteria algorithm produces an assignment of cost within $O(\log n)$ of the optimal while using no more than $O(\log n)$ copies of each bucket. The bicriteria result, which is based on rounding an LP (linear program) [27] can be considered the theoretical center-piece and contains the main ideas used in the other LP-based results in this paper.

[5] For time-varying requirements the problem can be modeled by #resources x #time-periods dimensions
[6] This is a modeling approximation and does not exactly capture 5 minute samples.



(a) VITA(min). The simplest unbounded case is inapproximable, and we give a bicriteria guarantee for the hardest case.

(b) VITA(max) and VITA(max − min). The unconstrained, cases are exactly solvable and we have tight logarithmic guarantees for the constrained unbounded case.

- VITA(max). Our results are summarized in Fig. 1b. Our results for VITA(max) also apply to VITA(max − min). We remind the reader that the unconstrained bounded box is empty because the algorithmic result for the harder unconstrained unbounded case (further up the lattice) applies .
- VITA(2^{nd} max). $2nd$ max turns out to be a particularly difficult problem from the standpoint of characterizing its computational complexity. We consider the unweighted (or uniform weights) unconstrained case and the requirement that the number of buckets exceeds the number of dimensions. With these restrictions we are able to demonstrate an LP-based approximation algorithm that achieves a logarithmic factor of approximation. We also show that unconstrained, bounded VITA(2^{nd} max) is weakly NP-hard [19].

This paper got its start in practical considerations at Nutanix - a leading hybrid cloud provider. Faced with a seeming plethora of different cloud colocation use-cases we wondered whether they could be tackled using a common approach. The VITA framework answers this question by providing a unified method for comparing against natural heuristics and a common basis for making pragmatic infrastructure decisions. We used real-world industrial traces from Nutanix, to conduct a detailed comparative analysis of the approximation algorithms, collectively dubbed *LP-Approx*, against 3 natural heuristics - *Conservative*, *Greedy* and *Local-Search*. *Conservative* treats each vector and its associated objective value in isolation. *Greedy* assigns vectors sequentially so as to minimize the increment in objective value. Working with a given assignment *Local-Search* swaps vectors when doing so improves the objective value. Our main finding is that from a practical standpoint too *LP-Approx* is the best in terms of solution-quality in all but one of the four cases (*Greedy* beats *LP-Approx* in the case of VITA(min)). Our work can serve as a valuable reminder of how principled and sophisticated techniques can often achieve superior quality on practical work-loads, while also providing theoretical guarantees.

1.3 Related Work

There is extensive theory literature on multidimensional versions of scheduling and packing problems. [11] is an informative survey that provides a variety of new results for multidimensional generalizations of three classical packing problems: multiprocessor scheduling, bin packing, and the knapsack problem. The vector scheduling problem seeks to schedule n d-dimensional tasks on m machines such that the maximum load over all dimensions and all machines is minimized. [11] provide a PTAS for the bounded dimensionality case and poly-logarithmic approximations for the unbounded case, improving upon [23]. For the vector bin packing problem (which seeks to minimize the number of bins needed to schedule all n tasks such that the maximum load on any dimension across all bins is bounded by a fixed quantity, say 1), they provide a logarithmic guarantee for the bounded dimensionality case, improving upon [14]. This result was subsequently further improved by [9]. A PTAS was provided for the multidimensional knapsack problem in the bounded dimension case by [18]. The key distinction between the vector scheduling problem of [11] and our framework is that they seek to minimize the maximum over the buckets and the dimensions whereas (in VITA(max)) we seek to maximize the weighted sum over buckets of the maximum dimension in each bucket. The multidimensional bin packing knapsack problems are capacitated whereas this paper deals with uncapacitated versions. There has also been a lot of work on geometric multidimensional packing where each vector is taken to represent a cuboid [13, 10]. To the best of our knowledge our VITA formulation is novel - surprising given its simplicity.

There is much recent literature (in conferences such as Euro-Par, ICDCS, SIGCOMM, CCGRID, IPDPS etc.,) substantiating the motivating scenarios we provide in the introduction (1.1) to this paper. We do not attempt to survey it in any meaningful way here. Peak provisioning and minimizing bottleneck usage is an area of active research in the systems community [12, 30]. Fairness in provisioning multi-dimensional resources is studied in [20]. The use of CSP (Constraint Satisfaction Programming) in placement has been investigated [22]. Energy considerations in placement have also been explored [16, 17] [15, 29, 28]. Building scalable systems that provide some guarantee against traffic analysis is an area of ongoing active research [32, 24, 26]. Relative to the specialized literature for each use-case our treatment is less nuanced (e.g., in reality, storage is less movable than compute, services are designed for (or to give the illusion of) continuous uptime, privacy is more subtle than just defeating traffic monitoring, etc). However, the generality of our approach enables us to abstract the essence of the different situations and apply sophisticated techniques from the theory of mathematical programming.

We present our results in the sections that follow. Section 2 presents results for linear F. Section 3 presents our results for VITA(min) while Section 4 contains our results for VITA(max) and VITA(max − min). VITA(2^{nd} max) results are presented in Section 5. Due to space constraints, all proofs are provided in the Appendix.

2 VITA(F) for linear F

By linear F we mean one of the following two situations:

- F is a vector and $F(\bar{V}) = \bar{F} \cdot \bar{V}$ (where we abuse notation slightly and use F as a function and a vector).
- F is a matrix and the weights are vectors with $*$ representing an inner-product so that $w_j * F$ is a scalar.

▶ **Lemma 1.** *VITA(F) can be solved exactly in polynomial time for linear F.*

▶ **Corollary 2.** *VITA(avg) can be computed exactly in polynomial time.*

Note that many real-world pricing situations are captured by linear F, such as charging separately for the usage of each resource (dimension).

3 VITA(min)

3.1 Unconstrained, Bounded - exact

First, we prove two lemmas about the optimal solution which will help us constrain the search space for our exact algorithm.

Without loss of generality assume that the bucket index j is sorted in order of increasing weight w_j.

▶ **Lemma 3.** *There exists an optimal solution which uses only the first b buckets, for $b \leq d$. Further, let $\min(j)$ be the dimension with the minimum value in bucket j; then, the set $\{\min(j)|1 \leq j \leq b\}$ has all distinct elements.*

We remind the reader that $\bar{V}_i(k)$ denotes the value in the k'th position of the vector $\bar{V}_i$.

▶ **Lemma 4.** *There exists an optimal solution in which item i is placed in that bucket j for which $w_j * V_i(\min(j))$ is minimized, amongst the first d buckets.*

The above two lemmas give rise to a straightforward search, Algorithm 1.

Algorithm 1 Exact Algorithm for Unconstrained Bounded VITA(min)

1: **for** each permutation Π of the first d buckets **do**
2: **for** each load vector $\bar{V}_i$ **do**
3: Place load vector $\bar{V}_i$ in that bucket j which minimizes $w_{\Pi(j)} * V_i(\min(\Pi(j)))$
4: Compute the value of the objective function for this permutation
5: Output the best value over all permutations and the corresponding assignment

▶ **Theorem 5.** *Unconstrained, Bounded VITA(min) can be computed exactly in time $O(m * n * d!)$*

3.2 Constrained, Bounded - strongly NP-hard

▶ **Theorem 6.** *Constrained, Bounded VITA(min) is strongly NP-hard.*

3.3 Unconstrained, Unbounded - inapproximable

▶ **Theorem 7.** *Unconstrained, Unbounded VITA(min) is inapproximable unless $P = NP$.*

3.4 Constrained, Unbounded - $O(\log n, \log n)$ bicriteria

Given that the problem is inapproximable (unless P=NP) we relax our expectations and settle for the next best kind of approximation - a bicriteria approximation, [25] where we relax not just the objective function but also the constraints. In this particular situation we will find a solution that uses at most $O(\log n)$ copies of each bucket while obtaining an assignment whose value is no worse than an $O(\log n)$ factor worse than the optimal solution which uses at most 1 copy of each bucket.

Consider the following LP (Linear Program). Let y_{jk} denote the fraction bucket j gives to dimension k, and x_{ijk} denote the weight vector i gives to dimension k of bucket j.

$$
\begin{aligned}
\min \sum_j w_j \sum_i \sum_k x_{ijk} v_{ik} & & \textbf{min-LP} \\
\text{s.t.} \sum_k y_{jk} &= 1 & \forall j \\
\sum_j y_{jk} &= 1 & \forall k \\
x_{ijk} &\leq y_{jk} & \forall i, j, k \\
\sum_j \sum_k x_{ijk} &\geq 1 & \forall i \\
x_{ijk} &\geq 0 & \forall i, j, k \\
y_{jk} &\geq 0 & \forall j, k
\end{aligned}
$$

▶ **Lemma 8.** *The above LP is a valid relaxation of Constrained, Unbounded VITA*(min)*.*

Let x^*_{ijk} and y^*_{jk} be the optimal solution of the LP. The algorithm is as follows.

Algorithm 2 Bicriteria Approximation for Constrained Bounded VITA(min)

for Each vector **do**
  Order its bucket-dimension pair by y^*_{jk} values. And maximize the corresponding x^*_{ijk} values in order. So there will be only one x^*_{ijk} value that is neither equal to y^*_{jk} nor 0.
  if This x^*_{ijk} value is greater or equal to $\frac{1}{2}y^*_{jk}$, **then**
    round it to y^*_{jk}
  else
    round it to 0, and double all the previous non-zero x^*_{ijk} values.
for $\ln \frac{n}{\varepsilon}$ times **do**
  for Each dimension k in each bucket j **do**
    With probability y^*_{jk} make a copy of bucket j in dimension k. And assign all the vectors with $x^*_{ijk} = y^*_{jk}$ to this bucket.

▶ **Theorem 9.** *Algorithm 2 is an* $O(\log n, \log n)$ *bicriteria approximation algorithm for Constrained Bounded VITA*(min)*.*

4 VITA(max)

Max - Min and Max are very similar, in that for the lower bound we can use the same log-hardness result since min is 0 and for the upper bound we can set the y variable to be greater than the difference of two dimensions for every pair of dimensions.

4.1 Unconstrained, Unbounded - exact

For example, unconstrained, bounded VITA(max) (see Fig. 1b) has an exact (polynomial-time) algorithm because a node above, namely unconstrained, unbounded VITA(max) does; further, this result is obviously tight and hence the square has a dotted background. Squares that do not have a dotted background represent open gaps that present opportunities for further research.

▶ **Theorem 10.** Unconstrained, Unbounded *VITA(max) can be computed exactly in time $O(m+n)$ time by placing all items into the bucket with the smallest weight.*

4.2 Constrained, Bounded - strongly NP-hard

▶ **Theorem 11.** Constrained, Bounded *VITA(max) is strongly NP-complete even when the number of dimension equals 2.*

4.3 Constrained, Unbounded - $\Theta(\log n)$

▶ **Lemma 12.** Constrained, Unbounded *VITA(max) is strongly NP-complete, and can not be approximated within $O(\log n)$.*

▶ **Lemma 13.** Constrained, Unbounded *VITA(max) is $O(\log n)$ approximable.*

Directly from Lemma 12 and 13, we get the following.

▶ **Corollary 14.** Constrained, Unbounded *VITA(max) is $\Theta(\log n)$ approximable.*

5 VITA($2^{nd}\max$)

We found VITA($2^{nd}\max$) to be a qualitatively harder problem and thus were forced to consider the restricted version where the weights are uniform and the number of buckets exceeds the (bounded) number of dimensions.

5.1 Unweighted, Bounded, Unconstrained - weakly NP-hard

▶ **Theorem 15.** *Bounded, Unconstrained VITA($2^{nd}\max$) is weakly NP-hard.*

5.2 Unweighted, Constrained, with number of buckets exceeding number of dimensions - $O(\log n)$ approximation

Consider the following LP. Let x_ij be the fraction of vector i assigned to bucket j.

$$\min \sum_{j=1}^{m} y_j \quad 2^{nd}\max - \text{LP}$$

$$\text{s.t.} \quad y_j \geq \sum_{i=1}^{n} x_{ij} \cdot v_{ik} \quad \forall j,k \ (j \neq k)$$

$$\sum_{j=1}^{m} x_{ij} \geq 1 \qquad \forall i$$

▶ **Lemma 16.** *The above LP $2^{nd}\max$-LP is a valid relaxation of constrained VITA($2^{nd}\max$) where the number of buckets exceeds the number of dimensions.*

▶ **Lemma 17.** *Unweighted, Constrained, VITA($2^{nd}\max$) with number of buckets exceeding number of dimensions can be approximated to factor $O(\log n)$.*

Proof. As with the algorithm and proof for min-LP, we need to repeat rounding $\{x_{ij}\}$ $O(\log n)$ times to make sure that all vectors are placed in some bucket with high probability. ◀

6 Experiments

We implemented *LP-Approx* and the three heuristics in Python, using Python 2.7.5. We use SageMath [31] and GLPK [8] as our Linear Programming Solver. We conducted our experiments on a single core of a 4-core Intel i7-3770 clocked at 3.4 GHz (0.25MB L2 cache per core, and 2MB L3 cache per core), with 16GiB of DDR3-1600 RAM.

Nutanix is a vendor of hyper-converged infrastructure appliances. For this paper we used a dataset obtained from an in-house cluster they maintain for testing and validation purposes. The cluster runs real customer workloads. The data was logged using the Prism system of Nutanix and then filtered, anonymized and aggregated before being handed to us. The dataset we received comprised of measurements logged every 5 mins of CPU, memory and storage used by 643 different services running continuously for the entire calendar month of August 2017. The data consisted of 643x8928 rows of 6 columns - timestamp, serviceID, CPU-usage, memory-usage, storage-utilization and bandwidth-usage.

The goal of our experiments was to compare the LP-based approximation algorithms to 3 natural polynomial-time heuristics - *Conservative*, *Greedy* and *Local-Search* - on each of the 4 problems - VITA(max), VITA(min), VITA($\max - \min$) and VITA($2^{nd}\max$). We briefly describe the 3 heuristics:

- *Conservative* This heuristic assigns each vector in isolation, i.e. it assigns each vector $\bar{V}_i$ to that bucket j which minimizes $w_j \cdot \tilde{F}(\bar{V}_i)$.
- *Greedy* The heuristic detailed in Algorithm 4 selects the vectors one by one in a random order and assigns to the bucket that minimizes the increase in the objective value.
- *Local-Search* Local search based vector placement in Algorithm 5 starts from a random feasible placement and repeatedly swaps vectors between two buckets to decrease the objective value. Since the size of the potential search space is exponential in n, the number of vectors, we restrict the heuristic to run the swapping step for a linear number of times.

It is easy to see that all the 3 heuristics can be arbitrarily bad ($\Omega(n)$) in terms of quality of approximation. However, we are interested in comparing their behavior on practical workloads vis a vis each other as well as the corresponding LP-based approximation algorithm. We run each of the 4 schemes (3 heuristics and 1 approximation algorithm) on samples of n vectors drawn from the dataset. Each sample is drawn uniformly from the entire dataset n runs from 10 to 100 in steps of 10. Given a sample we simulate each scheme on the sample to obtain a measure of the solution-quality and run-time[7]. For a given n we run as many samples as are needed to minimize the sample variance of the statistic (solution-quality or run-time) to below 1% of the sample mean. For VITA(max) we utilize the 3 dimensions - CPU, memory and storage - after a suitable normalization, and averaged over the entire month, i.e. we sample from 643 rows. For VITA(min) we aggregate CPU usage on an hourly basis (from the 5 minute measurements which reduces the dataset from 8928 to 744 rows per service). For VITA($\max - \min$) we aggregate bandwidth usage on an hourly basis per service. For VITA($2^{nd}\max$) we use the bandwidth usage on a 5min basis for each service. Based on our experiments we collected measurements on the two main considerations - (1) solution quality and, (2) running time, for each of VITA(max), VITA(min), VITA($\max - \min$) and

[7] We do not implement these schemes in the Nutanix system and then measure their performance as that would be expensive in terms of development time and would produce little additional clarity over the simulation based approach

VITA(2^{nd} max). In Figs. 2-5 we use VITA(f) in place *LP-Approx* to emphasize the specific function f under consideration.

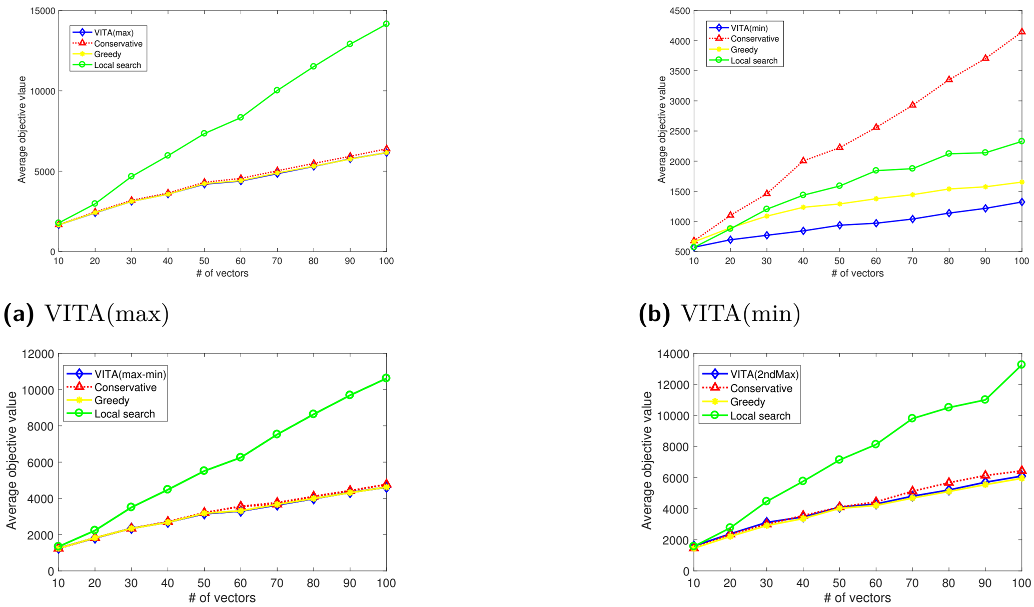
6.1 Solution quality

From Fig. 2a, Fig. 2c and Fig. 2d, it can be seen that the linear programming based approximation outperforms the heuristics for VITA(max), VITA(max − min) and VITA(2^{nd} max) by a factor of about 1.5. Unfortunately, the out-performance does not stand out visually because of the compression in the scale of the graph caused by the very poor performance of *Local-Search*. *Local-Search* performs particularly poorly in these 3 cases due to its dependence on the starting configuration.

For *minimizing the maintenance down time* in Fig. 2b, VITA(min) performs better than any of *Greedy*, *Local-Search* and *Conservative*. This is because VITA(min)'s bicriteria approximation scheme allows for the use of additional buckets, see Fig. 4. However, when the same number of extra buckets are given to all heuristics, we see that *Greedy* performs best.

6.2 Running time

Here we focus only on VITA and *Greedy* for two reasons: (1) Previous experiment results on solution quality show that VITA and *Greedy* are the two approaches of interest (2) *Local-Search* has much higher run time complexity than others. Fig. 3a-3d show that *Greedy*, basically linear-time, is superior to the LP based approximation algorithms.



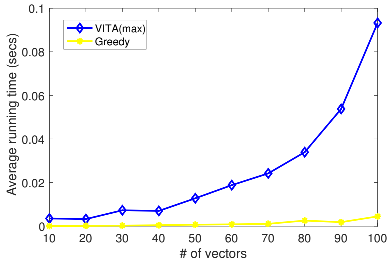
(a) VITA(max)

(b) VITA(min)

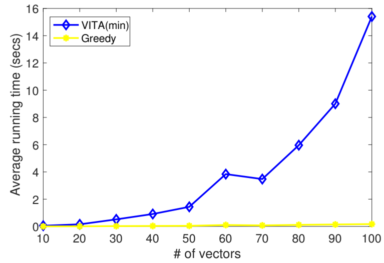
(c) VITA(max-min)

(d) VITA(2ndMax)

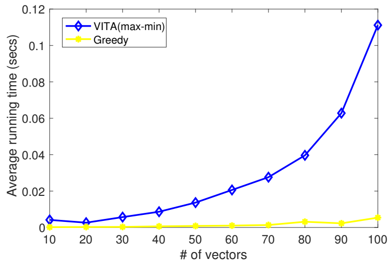
Figure 2 Quality of approximation of VITA(max, min, max-min, 2ndMax) vs {*Greedy*, *Conservative*, *Local-Search*}



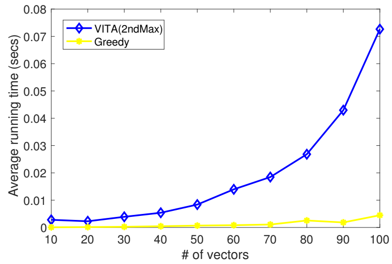
(a) VITA(max)



(b) VITA(min)



(c) VITA(max-min)



(d) VITA(2ndMax)

Figure 3 Running time of VITA(max, min, max-min, 2ndMax) vs *Greedy*

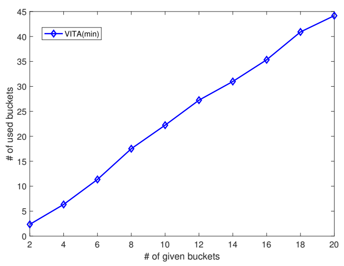


Figure 4 # of used buckets vs # of given buckets for VITA(min)

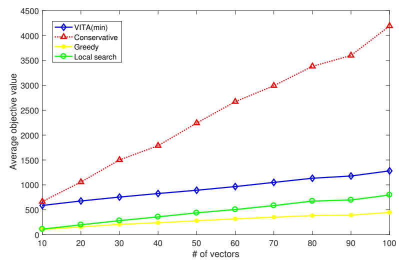


Figure 5 Solution quality with same number of additional buckets given to heuristics

7 Conclusion and Future work

We have proposed a new and general framework VITA that captures several naturally occurring problems in the context of hybrid clouds. We presented novel hardness results and approximation algorithms (using clever LP rounding). We conducted a detailed experimental evaluation comparing our approximation algorithm to several natural heuristics.

On the experimental side it would be interesting to characterize natural workloads and develop heuristics with provable (average-case) guarantees. Our theoretical work has left some obvious open gaps including constrained bounded VITA(min) and VITA(max) and removing the restrictions from our results for VITA(2^{nd} max). Another important direction for future investigation is devising distributed and online algorithms.

References

1 Amazon web services - cloud computing services. https://aws.amazon.com/.
2 Datadog - modern monitoring and analytics. https://www.datadoghq.com/.
3 Google cloud platform. https://cloud.google.com/.
4 Hewlett packard enterprise - hybrid it with cloud. https://www.hpe.com/us/en/home.html.
5 Microsoft azure cloud computing platform and services. https://azure.microsoft.com/en-us/.
6 Rightscale. https://www.rightscale.com/.
7 Turbonomic. https://turbonomic.com/.
8 GLPK (GNU linear programming kit), 2006. URL: http://www.gnu.org/software/glpk.
9 Nikhil Bansal, Alberto Caprara, and Maxim Sviridenko. A new approximation method for set covering problems, with applications to multidimensional bin packing. *SIAM J. Comput.*, 39(4):1256–1278, 2009.
10 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 13–25, 2014.
11 C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999. URL: citeseer.ist.psu.edu/chekuri99multidimensional.html.
12 Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 337–350, 2008.
13 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Multidimensional bin packing and other related problems: A survey. https://people.math.gatech.edu/~tetali/PUBLIS/CKPT.pdf.
14 W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.
15 Xenofontas A. Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Ph. Stoecklin. On the 95-percentile billing method. In *Passive and Active Network Measurement, 10th International Conference, PAM 2009, Seoul, Korea, April 1-3, 2009. Proceedings*, pages 207–216, 2009.
16 Corentin Dupont, Fabien Hermenier, Thomas Schulze, Robert Basmadjian, Andrey Somov, and Giovanni Giuliani. Plug4green: A flexible energy-aware VM manager to fit data centre particularities. *Ad Hoc Networks*, 25:505–519, 2015.
17 Corentin Dupont, Thomas Schulze, Giovanni Giuliani, Andrey Somov, and Fabien Hermenier. An energy aware framework for virtual machine placement in cloud federated data centres. In *e-Energy*, page 4. ACM, 2012.
18 A.M. Frieze and M. Clarke. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European J. Oper. Res.*, 15(1):100–109, 1984.
19 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
20 Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30 - April 1, 2011*, 2011.

21 Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, pages 23–27, San Jose, CA, 2013. USENIX. URL: `https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst`.

22 Fabien Hermenier, Julia L. Lawall, and Gilles Muller. Btrplace: A flexible consolidation manager for highly available applications. *IEEE Trans. Dependable Sec. Comput.*, 10(5):273–286, 2013.

23 D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987. URL: `citeseer.ist.psu.edu/470961.html`.

24 Rob Jansen, Kevin S. Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the tor network. In *5th Workshop on Cyber Security Experimentation and Test, CSET '12, Bellevue, WA, USA, August 6, 2012*, 2012.

25 Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Bicriteria network design problems. *J. Algorithms*, 28(1):142–171, 1998.

26 Nick Mathewson and Roger Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, pages 17–34, 2004.

27 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987. URL: `https://doi.org/10.1007/BF02579324`, `doi:10.1007/BF02579324`.

28 Vamseedhar Reddyvari Raja, Amogh Dhamdhere, Alessandra Scicchitano, Srinivas Shakkottai, kc claffy, and Simon Leinen. Volume-based transit pricing: Is 95 the right percentile? In *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings*, pages 77–87, 2014.

29 Vamseedhar Reddyvari Raja, Srinivas Shakkottai, Amogh Dhamdhere, and kc claffy. Fair, flexible and feasible ISP billing. *SIGMETRICS Performance Evaluation Review*, 42(3):25–28, 2014.

30 Mark Stillwell, Frédéric Vivien, and Henri Casanova. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 786–797, 2012.

31 The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.1)*, 2017. `http://www.sagemath.org`.

32 Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 137–152, 2015.

A Proofs

Proof of Lemma 1. Using the linearity of F the value of the objective function can be simplified thus

$$\sum_j w_j * F(\sum_{\bar{V}_i \in B_j} \bar{V}_i) = \sum_j \sum_{\bar{V}_i \in B_j} w_j * F(\bar{V}_i)$$

Hence minimizing the value of the objective function is simply a matter of finding the j that minimizes $w_j * F(\bar{V}_i)$ for each feasible $\bar{V}_i$. ◀

Proof of Corollary 2. Set $\bar{F} = [\frac{1}{d}, \frac{1}{d}, \ldots, \frac{1}{d}]$ where d is the dimension. It is straightforward to see that $\bar{F} \cdot \bar{V} = \frac{\sum_i V_i}{d}$. ◀

Proof of Lemma 3. It is clear that if in a solution two buckets have the same dimension with the minimum value then the bucket with the larger weight can be emptied into the smaller without increasing the value of the objective function. Thus the set of dimensions with the minimum value must be distinct across buckets and therefore the optimal solution need have at most d buckets. It is also clear that if the optimal solution does not involve a bucket j but does involve a bucket $j' > j$ then all the items in bucket j' can be moved to bucket j without increasing the value of the objective function. Thus the optimal solution may consist only of the first b buckets, for $b \leq d$. ◀

Proof of Lemma 4. Suppose not. Let item i be placed in bucket j'. Now if we move it to bucket j then the value of the objective function is changed by $-w_{j'} * V_i(\min(j')) + w_j * V_i(\min(j))$ which by definition is non-positive. Contradiction, and hence proved. ◀

Proof of Theorem 5. The correctness of Algorithm 1 follows from the prior two lemmas. The running time follows from the fact that the algorithm searches over $d!$ permutations and for each permutation it takes $O(m)$ time to assign each of the n load vectors. ◀

Proof of Theorem 6. The proof is by reduction from Bin Packing [19] which is strongly NP-hard. In an instance of Bin Packing we are given m bins of the same (constant) size S and a collection of n items a_i such that $\sum_i a_i = m * S$ and we need to decide if these n items can be packed into the m bins.

Given the instance of Bin Packing we create m buckets and $m + n$ load vectors of dimension 2. m of the load vectors are of the form $[S, 0]$ and the vectors are matched up with the buckets so that each such vector is necessarily assigned to its corresponding bucket. Then for each item a_i there is a load vector $[0, a_i]$ and these vectors are unconstrained and can be assigned to any bucket. All weights are set to 1. Now, it is easy to see that the given instance of Bin Packing is feasible if and only if the value of the objective function of VITA(min) is $m * S$. ◀

Proof of Theorem 7. The proof is by reduction from Set Cover [19].

In Set Cover we are given a collection of m sets over a universe of n elements and a number C and we need to decide whether there exists a subcollection of size C that covers all the elements.

We reduce the given instance of Set Cover to Unconstrained, Unbounded VITA(min) as follows: we let m be the dimension size as well as the number of buckets, one for each set. And, for each element i, we have an m-dimensional load vector:

$$\bar{V}_i(j) = \begin{cases} 1 & \text{if element } i \in \text{set } j \\ \infty & \text{otherwise} \end{cases}$$

We set the weights of C of the buckets to be 1 and the weights of the remaining buckets to be ∞.

It is easy to see that the value of the objective function for Unconstrained, Unbounded VITA(min) is C if and only if there exist C sets covering all the elements, otherwise the value of the objective function is ∞. Thus, Unconstrained, Unbounded VITA(min) cannot be approximated to any factor. ◀

Proof of Lemma 8. First we need to verify that this LP is a valid relaxation of the original problem. In other words, every solution of the original problem can be translated to the integer solution of this LP. And every integer solution of this LP is a valid solution of the original problem.

Suppose we have a solution of the original problem. Let $\min(j)$ be the minimum dimension of bucket j, and $\sigma(i)$ be the bucket assigned for load vector i. The value of the objective function for this solution is $\sum_j w_j \sum_{i:\sigma(i)=j} \bar{V}_i(\min(j))$. Now construct the integer solution of the LP. Let

$$y_{jk} = \begin{cases} 1 & \text{if } k = m(j) \\ 0 & \text{otherwise} \end{cases}$$

and

$$x_{ijk} = \begin{cases} 1 & \text{if } j = \sigma(i),\ k = m(j) \\ 0 & \text{otherwise} \end{cases}$$

Because each bucket only has one minimum dimension, the first constraint is satisfied. And each vector is assigned to one bucket, so the second and third constraints are satisfied also. On the other hand, if we have the integer solution, we can assign $\min(j) = k$ and $\sigma(i) = j$ to have a valid solution of the original problem. So there is a one to one relation between the integer solutions of the LP and the solutions of the original problem. Furthermore, the objective function of the LP is the same as the objective function of the original problem. So the optimal integer LP solution must map to the optimal solution of the original problem, and vice versa. ◀

Proof of Theorem 9. Notice that, in our algorithm we assume that $x^*_{ijk} = y^*_{jk}$ or 0. This is not hard to achieve. For each item, it will order its favorite bin-dimension pair by y^*_{jk} values. And maximize the corresponding x^*_{ijk} values in order. So there is only one x^*_{ijk} value that is not equal to y^*_{jk} value or 0. If this x^*_{ijk} value is greater or equal to $\frac{1}{2}y^*_{jk}$, we can round it to y^*_{jk}. Our new objective value is within twice the LP value. If not, we could round it to 0, and double all the previous non-zero x^*_{ijk} values. Then our value is still within twice the LP value. Even if we don't double the previous x^*_{ijk} values, we still have $\sum_{j,k} x^*_{ijk} \geq 1/2$, which we could use to bound the value output by our algorithm.

The expected value of the solution obtained by the (above randomized) Algorithm 2 is exactly the same as the optimum value of the LP. The expected number of copies of each bucket we make is $\sum_k y_{jk} = 1$. And the probability that vector i is not assigned to one of the buckets is: (where $s = m * d$),

$$\Pi_{j,k}(1 - x^*_{ijk}) \leq \left(1 - \frac{\sum_{j,k} x^*_{ijk}}{s}\right)^s = \left(1 - \frac{1}{s}\right)^s \leq e^{-1}$$

So, if we repeat for $t = \ln \frac{n}{\varepsilon}$ times, then

$$\begin{aligned} &Pr[\text{some vector is not assigned}] \\ &\leq \sum_i Pr[\text{vector } i \text{ is not assigned}] = \frac{n}{e^t} = \varepsilon \end{aligned}$$

The expected value of the solution is $OPT_{LP} \cdot \ln \frac{n}{\varepsilon}$. The expected number of copies of a bucket is $\ln \frac{n}{\varepsilon}$. Thus Algorithm 2 gives a $(\log n, \log n)$-approximation to Constrained Bounded VITA(min). ◀

Proof of Theorem 10. We first show that the bucket with the smallest weight will always be used in the optimal solution. If the bucket with smallest weight is not used in the optimal solution, we can always move all the items in one bucket with non-smallest weight to the bucket with the smallest weight to improve the solution.

Now, we show that if we move all the items in the buckets with non-smallest weight to the bucket with smallest weight, the objective value of this new solution will not increase.

To see this, let the bucket $B_0$with the smallest weight w_0. Let the aggregated vector in B_0 be $\bar{V}_0$. Let the bucket B_i with a non-smallest weight w_i in the solution, the aggregated vector in B_i be $\bar{V}_i$.

It is easy to see that $w_0 \cdot \max(\bar{V}_0 + \bar{V}_i) \leq w_0 \cdot (\max(\bar{V}_0) + \max(\bar{V}_1)) \leq w_0 \cdot \max(\bar{V}_0) + w_i \cdot \max(\bar{V}_i)$.

Thus, moving all items from B_i to B_0 will not increase the objective value of the current solution.

Moving all items to the smallest weighted buckets is optimal. ◀

Proof of Theorem 11. We prove by making reduction from bin packing. For k bins with capacity c, we correspondingly assign k buckets. As part of input vectors, we will have k 2-dimensional vectors $(c, 0)$. Each of them are strictly constrained to each bucket. Then for each item i with size s_i in the problem of bin packing, we create a 2-dimensional vector $(0, s_i)$ which can be put into any bucket. We further let each bucket have uniform weight of 1. Then there exists k bins that can hold all the items in the bin packing problem if and only if the objective value of this VITA(max) that equals kc is reachable. ◀

Proof of Lemma 12. We prove by making reduction from set cover. First we let the number of dimensions of input vector in VITA(max) be the number of elements in the set cover problem. For each element $s_i (i = 1 \sim n)$, we correspondingly let vector $\bar{V}_i$ has value one on dimension i, has value zero on all the other dimensions. Thus, there are no two element vectors has one value on the same dimension.

Each subset S_j maps to a bucket B_j. If element $s_i \in S_j$, then $\bar{V}_i$ can be placed at bucket B_j.

Thus, there exists k subsets that cover all the elements if and only if the objective value of this VITA(max) that equals k is reachable. ◀

Proof of Lemma 13. Consider the following LP. Let x_{ij} be the fraction of item i assigned to bucket j.

$$\min \sum_{j=1}^{m} w_j * y_j \quad \max -\text{LP}$$

$$\text{s.t.} \quad y_j \geq \sum_{i=1}^{n} x_{ij} \cdot v_{ik} \qquad \forall j, k$$

$$\sum_{j=1}^{m} x_{ij} \geq 1 \qquad \forall i$$

It is easy to see that this max-LP is a valid relaxation of *constrained*, *unbounded* VITA(max). Then we need to repeat rounding $\{x_{ij}\}$ $O(\log n)$ times to make sure that all items are placed to some buckets with high probability. The proof is similar to the part in min-LP. ◀

Proof of Theorem 15. The proof is by reduction from Partition [19]. In an instance of Partition we are given an array of numbers $a_1, a_2, \ldots, a_n$ such that $\sum_{i=1}^{n} a_i = 2B$, and we are required to decide whether there exist a partition of these numbers into two subsets such that the sum of numbers in each subset is B.

Given an instance of Partition we reduce it to an instance of Bounded, Constrained VITA(2^{nd} max) as follows: our reduction will use 3 dimensions. For each number a_i we construct the load vector $[0, 0, a_i]$. We add another two vectors, $[L, B, 0]$ and $[B, L, 0]$, where $L >> B$, to the collection of vectors. And, there are two (3-dimensional) buckets with uniform weights which we take to be 1. In an optimal assignment vectors $[L, B, 0]$ and $[B, L, 0]$ will be assigned to different buckets because $L >> B$. Thus, the contribution of each bucket is at least B and the value of the objective function is always at least $2B$. Now, from our construction, it is easy to see that if the given instance of Partition has a partition into two subsets with equal sums then the value of the objective function (of the instance) of VITA(2^{nd} max) (to which it is reduced) is $2B$. And if there is no equal sum partition into two subsets, then one of the buckets necessarily has a 2^{nd} max dimension value greater than B, which means that the objective value has to be larger than $2B$. ◀

Proof of 16. First we need to verify that y_j really represents the 2^{nd}-maximum dimension in the LP solution. From the first LP constraint, we know y_j is either the maximum dimension or the 2^{nd}-maximum dimension. The following proof shows that based on the current LP optimum we could come up with a new LP optimum solution in which y_j is the 2^{nd}-maximum dimension of bin j. For each bin j with y_j as maximum dimension, there are only 2 cases, as follows.

Case 1: the item, with y_j's corresponding dimension as "free" dimension, has its "free" dimension as maximum. In bin j the "free" dimension is j^{th} dimension. Assume y_j represents the value in dimension d_j of bin j, then we can find the bin in which dimension d_j is the maximum ("free" dimension). Merge these two bins together and set d_j as the "free" dimension of this bin. In the new solution, the cost won't be more than the previous optimal solution, which means this is also an optimal solution.

Case 2: the item, with y_j's corresponding dimension as "free" dimension, doesn't have its "free" dimension as maximum. Let bin j have "free" dimension j. y_j represents the value of dimension d_j of bin j and it is the maximum dimension. Bin k has d_j as "free" dimension. And y_k is the maximum dimension of bin k. Then swap these two bins. The cost of new bin k is less than y_j and the cost of new bin j is at most equal to y_k. So the cost of new solution is better than the original optimal solution. This is a contradiction, which means this case couldn't happen.

To sum up, given an optimal solution of the LP, we can come up a new optimal solution in which each y_j represents the 2^{nd}-maximum dimension of bin j. ◀

B Experiment

B.1 Pseudo-code of heuristics

B.2 Histograms of requests

Algorithm 3 Heuristic 1 - Conservative

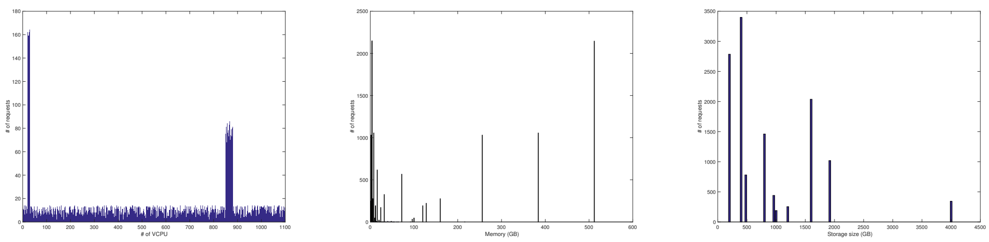
for each vector **do**
    Assign the vector V_i to that bucket j which minimizes $w \cdot F(V_i)$.

Algorithm 4 Heuristic 2 - Greedy

Shuffle the order of vectors;
for each vector **do**
    Assign the vector to that bucket such that the current objective value is raised the least;

Algorithm 5 Heuristic 3 - Local-Search

for each vector **do**
    Randomly assign it to a feasible bucket by affinity constraint;
for 1 to $poly(n)$ steps **do**
    for every two buckets **do**
        Swap any pair of two vectors if the swap will reduce the objective value;



(a) # of VCPUs **(b)** Memory size **(c)** Storage

Figure 6 Histograms of requested resources