High Availability for VM Placement and a Stochastic Model for Multiple Knapsack

Bochao Shen[1], Ravi Sundaram[1], Alexander Russell[2], Srinivas Aiyar[3]
Karan Gupta[3], Abhinay Nagpal[3], Aditya Ramesh[3], Himanshu Shukla[3]
[1]College of Computer and Information Science, Northeastern University, Boston, MA, USA
[2] Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, USA
[3] Nutanix, Inc., San Jose, CA, USA
Email: [1]{ordinary, koods}@ccs.neu.edu, [2]acr@cse.uconn.edu
[3]{sriniva.aiyar, karan.gupta, anagpa, aramesh, hshukla}@nutanix.com

***Abstract*—k-HA (high-Availability) is an important fault-tolerance property of VM placement in clouds and clusters - it is the ability to tolerate up to k host failures by relocating VMs from failed hosts without disrupting other VMs. It has long been assumed [1] that deciding the existence of a k-HA placement is Σ_3^P-hard. In a surprising yet simple result we show that k-HA reduces to multiple knapsack and hence is in NP$= \Sigma_1^P$.**

We propose a stochastic model for multiple knapsack that not only captures real-world workloads but also provides a uniform basis for comparing the efficiencies of different polynomial-time heuristics. We prove, using the central limit theorem and linear programming, that, there exists a *best* polynomial-time heuristic, albeit impractical from the standpoint of implementation.

We turn to industry practice and discuss the drawbacks of commonly used heuristics - *First-fit*, *Best-fit*, *Worst-fit*, *MTHM* and *CSP*. Load-balancing is a fundamental customer requirement in industry. Based on a large real-world dataset of cluster workloads (from industry leader Nutanix) we show that the natural load-balancing heuristic - *Water-filling* - has several excellent properties. We compare and contrast *Water-filling* with *MTHM* using our stochastic model and find that *Water-filling* is a heuristic of choice.

***Keywords*-Virtual Machine Placement, High Availability; Multiple Knapsack Problem**

I. Motivation

Cloud computing has established itself as a mainstay of modern computing infrastructures. Clouds enable the efficient utilization of resources on an as-needed basis by dynamically configuring these resources to accommodate varying workload needs. The core technology at the heart of public cloud datacenters and private clusters is *virtualization*. Virtualization is the use of shared resources to create and operate virtual machines (VMs) [2]; a VM is an operating system or software that emulates the behavior of a computing system with a specified set of resource characteristics, such as CPU and memory capacity. Virtualization allows for the execution of an application onto heterogeneous systems as well as multiple applications in parallel. Virtualization also enables live migration or the movement of running VMs between hosts. The enormous flexibility afforded by virtualization enables the placement (mapping of VMs to physical hosts) and rebalancing of VMs for a variety of reasons including performance and cost-efficient resource utilization [3]. In this paper we will primarily be concerned with fault-tolerance, i.e. robustness to failures.

An important aspect of fault-tolerance in a cluster is High Availability (HA) [4]. In general, HA is the property of ensuring continuity of services (applications) despite the failures of hosts, VMs or the applications themselves. In the context of this paper we will interpret HA narrowly as the ability to tolerate host failures by restarting the VMs (from the failed hosts) in back up hosts without affecting any other existing running VMs. We define k-HA to be the property of a placement to tolerate the failure of up to k hosts (sequentially or in parallel). In practice the response to a failure or multiple failures the system could involve restarting of the failed VMs in other hosts; in platforms with HwPFA (Hardware Predicted Failure Analysis alerts it could also involve live migration of the VMs in advance of an impending failure. In either case the key algorithmic requirement is ensuring the sufficiency of resources in the back up hosts to support the new VMs. In general resources are multi-dimensional (such as CPU, memory, IO etc) but in this paper we focus on a single resource - namely, memory.

HA has been studied extensively both in the algorithms and the systems communities. Unfortunately, even the special of case of 0-HA with a single resource is the probem of bin packing and hence already NP-complete [5]. Partly as a reaction to the hardness of HA the systems community has turned to polynomial-time heuristics on the one hand or AI (artificial intelligence) and CSP (Constraint Satisfaction Programming) on the other. Heuristics, the choice of industry, are typically quick to return some solution but fail to provide guarantees on the quality of the solution. AI and CSP, areas of academic research, allow much greater expressiveness involving additional constraints such as affinity and co-location requirements as well as multiple resource dimensions. However, both of these techniques are very slow, particularly CSP which solves SAT (satisfiability, an NP-complete problem [5]) at its core.

In spite of the attention that HA has received the fundamental theoretical questions still remain unanswered: Given VMs (with sizes) and hosts (with capacities) and k how easy/hard is it to decide whether there exists a feasible k-HA placement? Given a placement and k how easy/hard is it

978-1-5090-2991-4/17/$31.00 ©2017 IEEE

to decide whether it (the given placement) is k-HA? And, on the practical side, the fundamental questions are: Given that typical inputs encountered in practice are not adversarially chosen (worst-case) how do we meaningfully compare the quality of different heuristics? Given that load-balancing is a basic customer requirement what is the best heuristic to use? In this paper we make substantial progress towards answering such questions.

II. Our Contributions

Our main technical contributions are as follows:

- **k-HA is NP-complete** Counter to intuition we prove that deciding the *existence* of a k-HA placement is NP-complete by a simple reduction to MKP (Multiple Knapsack Problem) [6]. We note two things here 1. the clever part of the proof is in showing that k-HA is in NP and not that it is NP-hard, which is why the reduction is to (and not from) MKP, and 2. we do, however, believe that the problem of deciding whether a *given* placement is k-HA is Π_2^P-complete.
- **IID-IK and existence of a best heuristic** We propose a simple stochastic model, IID-IK (Independently and Identically Distributed Items and Knapsacks), for MKP, that captures the essence of real-world workloads as well as synthetic generators. We define an asymptotic metric $\mathbb{PE}$ for comparing the quality of different heuristics. We prove that the quality of the *best* heuristic is computable.
- **Water-filling is the best** We discuss the shortcomings of some heuristics used in practice- *First-fit*, *Best-fit*, *Worst-fit*, *CSP* and the industry leader *MTHM*. Based on a dataset of cluster workloads from Nutanix we posit two natural subcases: the *doubling world* where VM sizes are powers of 2 (this is indeed the case for Amazon's AWS where VM sizes are restricted to be powers of 2) and the *gold-dust world* where VM sizes are much smaller than host capacities. We consider *Water-filling*, which smooths out the load by definition, and show that it is optimal for the doubling world and near-optimal for the gold-dust world. We also show that *Water-filling* provides a simple and easily computed criterion for determining the largest k for which a given placement is k-HA. We also compare *Water-filling* and MTHM using IID-IK with distributions instantiated from a large real-world dataset of cluster workloads provided by Nutanix Inc. We conclude that *Water-filling* is the heuristic of choice, from among the heuristics in consideration.

On a conceptual level we have shown that deciding the existence of a k-HA placement is easier than checking whether a given placement is k-HA. We have presented a simple stochastic model for workload generation that is both useful for comparing heuristics and tractable. We have shown, through a variety of arguments, that *Water-filling* satisfies customer requirement for load-balancing while possessing useful k-HA properties.

III. Related Work

[4] is a comprehensive source on a variety of techniques leveraging virtualization for HA including replication patterns such as active/active, active/passive and cold standby. Our work derived inspiration from [1] that formally defines the k-HA problem - they call it k-resilient HA. They observe that k-HA lends itself naturally to expression as a statement in Second Order Logic and present a transformation as a statement in First-Order Logic thus allowing the constraints to be solved by a generic CSP solver. However, this transformation is not exact in that there may be solutions to the original problem even though the transformed input to the CSP solver may have none. We note that they extend the notion of k-HA to VMs - a k-HA VM must be tolerant to the failure of up to k hosts through relocation. Our result can be seen as an exact representation of k-HA in First Order Logic, though our result does not apply to the notion of k-HA extended to VMs. We represent our results in the language of complexity theory rather than in terms of logic systems, which is the more accurate way to capture computational issues. [7] describes BtrPlace, a CSP solver customized for handling VM placements that allows the user considerable expressiveness in specifying custom requirements. Though BtrPlace is generally fast and scalable, the CSP solver is slow to check feasibility of HA requirements. [8] consider the related problem of replica VM placement constrained by bounded communication delays between the replicas. They prove the NP-completeness of their problem and compare a variety of different heuristics. Replica placement is different from the strategy of relocating VMs from failed hosts to other hosts considered in this paper.

From a systems perspective some related works include [9] which describes an HA-aware scheduler based on openstack, [10] that considers the issue of hardware redundancy, [11] which proposes an availability-aware scheduler that dynamically manages computing resources based on user-specified requirements and [12] that describes a method for related VMs to share check point images.

In this paper we show that the problem of VM placement reduces to MKP (Multiple Knapsack Problem) an NP-complete problem [5]. [13] presents an EPTAS for MKP. Our stochastic model, IID-IK is inspired by the model for bin packing investigated in [14] where items are drawn from a continuous distribution. Subsequent to [14] there has been much work on distributions with both continuous and discrete supports [15], [16]. [17] is a recent work with several related references. our model IID-IK assumes a discrete support which we also refer to as a finite support since they are the same for a bounded domain. *MTHM* is a polynomial-time heuristic for MKP presented in [6] and deployed in some commercial providers of cloud infrastructure.

In Section IV we state the k-HA VM placement problem. Section V proves k-HA placement is NP-complete. We propose our stochastic model IID-IK model for multiple knapsack in Section VI. Section VII studies the performance of different heuristics and points out that *Water-filling* is the heuristic of

choice. Our paper concludes in Section VIII.

IV. Problem statement and Preliminaries

A. VMPP and VMPP-AC

We start by defining the most basic VM placement problem (*V*MPP): Let V denote the set of all n VMs and H denote the set of all m hosts. Each VM v_j requires s_j (GB) sized memory to run. Each host h_i has a total memory capacity c_i (GB) configured for VMs. To each VM v_j is attached profit p_j which reflects the value or reward for placing VMs. The goal is to place VMs on hosts, so as to maximize the sum of the profits of the placed VMs, while satisfying the constraint that for every host the sum of the memories of the VMs placed on it must not exceed its capacity, i.e., $\forall_i \sum_{v_j \in h_i} s_j \leq c_i$.

It is clear that VMPP is just MKP [13], [5] and therefore NP-complete. For the rest of this paper we will assume that all profits p_j are the same and hence the goal of VMPP is to place the maximum number of VMs on the hosts. This special version of VMPP (or MKP) continues to be NP-complete, in fact strongly NP-complete as bin packing reduces to it.

At this point the reader might wonder why we connected VMPP to MKP rather than to bin packing - the reason is that in the bin packing problem the bins are of uniform capacity whereas in VMPP hosts can have different capacities just as knapsacks in MKP. As decision problems bin packing and MKP with uniform profits are the same in complexity but as optimization problems they are different - in bin packing the items are given and the goal is to minimize the number of bins needed to place all of the items whereas in MKP both the items and knapsacks are given and the goal is to to maximize the number of placed items without violating the capacity constraints of any host. Thus MKP is in concordance with VMPP where, too, the VMs and hosts are given.

In the rest of this subsection we consider VMPP-AC (the VM placement problem with affinity constraints). This problem is a digression in that we do not use VMPP-AC in the rest of the paper, nevertheless the placement of uniformly sized VMs with affinity constraints was mentioned in [1] as a problem with undetermined complexity. We resolve its complexity in this subsection.

Three kinds of affinity constraints are mentioned in [1] (1) VM-VM co-location constraint: e.g. v_i and v_j must always be placed on the same host (in order to minimize the communication cost, etc); (2) VM-VM anti co-location constraint: e.g. v_i and v_j must always be placed on different hosts (so as to be separate from each other due to privacy or security reasons); (3) VM-host affinity constraint: e.g. v_i can only be placed on a specific subset of hosts.

Theorem 1: VMPP-AC with uniform VM sizes is NP-complete.

Proof: We give a reduction from 3DM (3D-matching) which is NP-hard [5].

The input in 3DM is a tripartite graph with each partition having n vertices. The question is whether the edges of the graph can be partitioned into exactly n triangles.

We transform a given instance of 3D-Matching into an instance of VMPP-AC as follows: let the tripartite graph be $G(\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{E})$, where $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ denote the three partitions of vertices, and $\mathcal{E}$ denotes the edges crossing them. We interpret the three partitions as follows - the set of VMS V = $\mathcal{V}_1 \cup \mathcal{V}_2$ and the set of hosts $H = \mathcal{V}_3$. We interpret the edge (v_s, v_t) between $v_s \in \mathcal{V}_1$ and $v_t \in \mathcal{V}_2$ to mean that v_s and v_t can be placed together. (Note that these constraints are easily expressed using VM-VM anti co-location constraints.) The edge (v_s, h_i) between $v_s \in \mathcal{V}_1$ or $\mathcal{V}_2$ and $h_i \in \mathcal{V}_3$ means v_s can be placed on h_i. (Note that these constraints are easily expressed using VM-host affinity constraints.) We set the size of each VM to be 1 and the capacity of each host to be 2. It is easy to see that there exists a partition of the edges of the tripartite graph into n triangles iff all the VMs can be packed. ■

B. Exists?k-HA and Is?k-HA

The HA property of a VM placement guarantees the system can tolerate some level of failure in hosts. Basically, HA should satisfy the following two conditions: (1) VMs on failed hosts should be re-instantiated quickly in other running hosts; (2) VMs originally on those running hosts should not be affected, i.e. suspended, migrated, etc, at all.

This adds additional complexity in the VM placement problem. When VMs are initially placed, one has to consider if there will be enough space on the still running hosts for the VMs assigned to a failed host.

The k-HA property of a VM placement can quantify such fault tolerance. Specifically, a placement is said to have the k-HA property, if for all sequences of up to k host failures there is enough memory space on the running hosts for the VMs from the failed hosts to be re-instantiated (on the running hosts).

We define two decision problems: Exists?k-HA and Is?k-HA. Exists?k-HA is the problem of deciding whether a given instance of VMPP has any placement with the k-HA property. Is?k-HA is the problem of deciding whether a given placement has the k-HA property.

Note that VMPP is the same as Exists?0-HA. Thus, we can say that deciding the existence of a k-HA placement is at least as hard as the original VM placement problem.

V. k-HA is NP-complete

In a counter-intuitive result we show that the HA problem can be reduced to multiple knapsack and hence is NP-complete.

A. Exists?k-HA is NP-hard

Naively, one would expect that Exists?k-HA lies in the third level of the polynomial hierarchy, Σ_3^P [18]. This is because it would seem to require 3 alternating quantifiers in the following form:

there-exists placement *for-all* upto k host failures *there-exists* rebalancing *such-that* the original placement and rebalancing strategy are valid.

However, we show that in fact it lies in $NP = \Sigma_1^P$.

Theorem 2: For all k, Exists?k-HA reduces to MKP.

Proof: Given m knapsacks and n items pack the items into the smallest $m-k$ knapsacks, i.e. set aside the k largest capacity knapsacks and pack the items into the rest. The given system of knapsacks and items is k-HA iff this packing is achievable. Clearly, if the system is k-HA then it should be packable into the smallest $m-k$ knapsacks in order to be resilient to the failure scenario where the k largest knapsacks fail. And conversely, the packing into the $m-k$ smallest knapsacks can tolerate failure of any k knapsacks - if one of the smaller knapsacks fails then move its items into one of the larger empty knapsacks and there will always be at least one larger empty knapsack since we started out with k empty knapsacks. ∎

The key point to note is that MKP (or VMPP) is equivalent to Exists?0-HA, hence it directly follows that Exists?k-HA is at least NP-hard, and so the clever part of the above theorem is in showing that Exists?k-HA is no harder by reducing it to MKP.

Corollary 1: Exists?k-HA in NP-complete.

Note that even though Exists?k-HA suddenly seems to have become easy we still believe that Is?k-HA continues to be hard, i.e. Π_2^P-complete, though we are unable to prove it yet.

VI. IID-IK AND BEST HEURISTIC

We propose a stochastic model for multiple knapsack that not only captures real-world workloads but also provides a uniform basis for comparing the performance of different polynomial-time heuristics. We prove that, in a fairly general sub-case, there is a *best* heuristic in our model, i.e., a heuristic with optimal performance.

Having shown that HA reduces to the multiple knapsack problem which is NP-complete we understand that we are limited to using polynomial-time heuristics. The question we are faced with is which heuristic to use, i.e. how to measure the performance or efficiency of heuristics. One standard way to do this is to look at their quality of approximation. However, even quality of approximation is a worst-case measure. We argue that workloads drawn in practice have a lot more regularity. In fact, based on a large dataset of client workloads drawn from the customer-base of one of the largest providers of private cloud infrastructure we find that workloads can be characterized as independent draws from distributions for knapsacks and items. Inspired by this finding as well as by an early analysis of bin packing [14] we propose the following natural stochastic model of multiple knapsack:

Items are drawn independently from a distribution of items - the item distribution specifies both the size and the value of the draw ; knapsacks are drawn independently from a distribution of knapsacks - the knapsack distribution specifies the size of the knapsack. We assume that distributions for both knapsacks and items have finite support, i.e., there are only a finite number of different types of items and knapsacks. For simplicity we assume that all items have value 1 - our definitions and results can easily be generalized to the case with arbitrary values.

This model gives us a solid mathematical basis for comparing different heuristics. We now describe the measure $\mathbb{PE}(H)$ which captures the asymptotic packing efficiency of the heuristic H. Draw n samples from the item distribution; then keep drawing samples one-by-one from the knapsack distribution; after each knapsack draw check whether H packs all the items into the drawn collection of knapsacks; let $\min_H(n)$ denote the number of drawn knapsacks when H successfully packs all items for the first time. Define the finite packing efficiency for n:

$$\mathbb{PE}(H,n) \triangleq \frac{n}{\min_H(n)}$$

Define the *packing efficiency* of H to be:

$$\mathbb{PE}(H) = \liminf_{n\to\infty} \mathbb{PE}(H,n)$$

$\mathbb{PE}(H)$ is well-defined since $\limsup$ exists for all bounded sequences and is the expected number of items the heuristic packs per bin, in the asymptotic limit. We also note that the definition of $\mathbb{PE}$ is robust: instead of drawing bins until we can pack the n items we could also define it in terms of the process where we draw items until they will no longer fit inside the m bins.

Through simulations and for simple distributions we are able to verify the existence of packing efficiencies for *Water-filling* and *MTHM*. We are led to the natural question of whether, given the item and knapsack distributions, there exists a *best* heuristic and surprisingly we can show that the answer is yes for distributions with a finite support, which captures all real-world situations. Informally the theorem below states that in the stochastic model of multiple knapsack where the distributions have finite support, we can find the best heuristic and its packing efficiency (which is optimal).

Theorem 3: There exists a heuristic $\mathcal{H}$ that achieves the maximum possible $\mathbb{PE}$ in the IID-IK model; both $\mathcal{H}$ and $\mathbb{PE}(\mathcal{H})$ are explicitly computable.

Proof: We first give the intuition behind the proof to enable the reader to follow the formal argument easily.

That the distributions have finite support means there are a finite number of different types of item sizes and knapsack capacities. Further, by the Central Limit Theorem when the number of items and knapsacks goes to infinity then we will get close to the expected proportion of each size (for items) and capacity (for knapsacks). Now consider all possible ways of packing each of these different knapsacks with different combinations of items - there are only a finite number of different packed knapsack configurations. (This includes all such configurations, including for example the empty knapsack). With one variable per packed knapsack configuration representing its proportion in the final solution we can write a linear program to maximize packing efficiency constraining the proportions of items sizes and knapsack capacities to fit the proportions dictated by the Central Limit Theorem. Thus the maximum packing efficiency can be calculated in time

polynomial in the support size. Formally, let ι and κ index the finite support set of items and knapsacks respectively. Let the ι'th item be generated in proportion P_ι and let the κ'th knapsack be generated in proportion P_κ. Let $w(\kappa)$ denote the number of distinct ways that the κ'th knapsack can be packed with items, and let ω be the corresponding index. Let $N_\kappa^\omega(\iota)$ denote the number of copies of item ι in the ω'th packing of the κ'th knapsack. Variable p_κ^ω is the proportion of the ω'th packing of the κ'th knapsack in the optimal packing. Variable pe is the packing efficiency that we are trying to maximize. Then the solution to the following LP gives the maximum possible $\mathbb{PE}$ in the IID-IK model:

$$\begin{array}{lll} \text{maximize} & pe & \\ \text{subject to} & \sum_{\omega,\kappa} p_\kappa^\omega (\sum_\iota N_\kappa^\omega(\iota)) = pe & \\ \forall_\iota & \sum_{\omega,kappa} p_\kappa^\omega N_\kappa^\omega(\iota) & = P_\kappa * pe \\ \forall_\kappa & \sum_\omega p_\kappa^\omega & = P_\kappa \end{array}$$

Since the support set is finite the above LP calculates the optimal $\mathbb{PE}$ in constant time. The optimal heuristic $\mathcal{H}$ is also derivable in straightforward fashion from the values of the p_κ^ω variables in the optimal solution: given any input the optimal heuristic is to utilize the ω'th packing of the κ'th knapsack to the extent of $\frac{n}{\mathbb{PE}(\mathcal{H})} * p_\kappa^\omega$ copies; any items left over can be given their own individual knapsack; by Chernoff bounds we are guaranteed that there will be at most $O(n^{0.5})$ additional knapsacks used so that asymptotically the packing efficiency will limit to $\mathbb{PE}$ since packing efficiency is a ratio which has $\min_{\mathcal{H}}(n) = \Omega(n)$ in the denominator. ■

Note that $\mathbb{PE}(\mathcal{H})$ is the best packing efficiency achievable when both items and knapsacks are generated using the stochastic model. However, given an arbitrary collection of items, we can potentially achieve better packing efficiencies if we are allowed to select the knapsacks. In fact, using an LP similar to the one above and the fact that fixed dimension integer programming is in P [19] it is possible to compute the best packing in time polynomial in the number of items.

VII. Analysis of heuristics

Given the reduction of k-HA to Multiple Knapsack Problem, k-HA is NP-complete. Unless $P = NP$, to compute k-HA placement efficiently in polynomial time is impossible. Thus, to solve a placement problem, we consider *CSP* and the following 5 polynomial time heuristics commonly used in industrial practice: *First-fit*, *Best-fit*, *Worst-fit*, *MTHM* and *Water-filling*. Each of these 5 heuristics is a different variant of greedy algorithm.

In our scenario of VM placement, the hosts are sorted in increasing order of size before being input into these heuristics. Given a sequence of VMs, *First-fit* and *Best-fit* will pack VMs one by one until all the hosts can hold no more. At each step, *First-fit* will put the VM into the smallest indexed host that has enough space, while *Best-fit* will put the VM into the tightest host that has enough space. For *Worst-fit*, it puts VMs one by one into the emptiest host until no additional VM can fit into any of the hosts.

MTHM is more complicated and we will give a more detailed description in the next subsection.

Water-filling first sorts the VM in decreasing order of size, then puts each VM into the lightest loaded VM that also has enough space for that VM.

Load-balancing, i.e., ensuring the load is evenly spread among the hosts, is a critical property desired by cloud clients. It is also a research topic that attracts much attention [20]. However, *First-fit*, *Best-fit* and *Worst-fit* will overload some hosts with many VMs while underloading other hosts with too few VMs.

Also, the time to compute a placement is critical both for initial placement of a set of VMs as well as contingency placement of VMs from a failed host since cloud providers are contracted to provide service with interruption/delay below some small threshold.

This makes a *CSP* solver impractical for industrial use. For example, the duration of a placement computation for 32 hosts can take on the order of minutes [1]. But industrial requirement demands the computation be at most on the order of milliseconds.

This leaves us with just two methods - *Water-filling* and *MTHM* - to consider. In the following subsections, we first describe how *MTHM* works and use a simple example to show the limitation of *MTHM*. Then we show that, in the *Gold-dust World*, *Water-filling* performs very close to the optimal. We also develop simple bounds for the level of HA achievable in the *Gold-dust World* - these bounds apply to all greedy variants though we state and prove them only for *Water-filling*. We further show that in the *Doubling World Water-filling* actually achieves optimality.

In the end of this section, we evaluate the relative performance between *Water-filling* and *MTHM* and conclude that *Water-filling* is the heuristic of choice.

A. MTHM revisited

We review a heuristic for Multiple Knapsack Problem – *MTHM* by Martello and Toth [6]. We first go over the heuristic at a high level, then we use a small example to show the limit of MTHM.

1) A high level overview of MTHM: MTHM takes the input: (a) descending sorted list of items in unit profit (value over weight); (b) ascending sorted list of knapsacks in sizes. It outputs the approximated maximized total value of packable items with the information of which knapsack to pack each of them into.

In the case of our VM placement, we assign all the items with uniform value 1. This reduces the goal to maximizing the number of the VMs that get placed. We just need to input a sorted VM list in order of increasing size.

The overall procedure of *MTHM* is:

i) (Initialization): For each knapsack, run the greedy step in Algorithm 1, i.e. place as many items from the

Algorithm 1 Greedy packing on one bin in MTHM

Given one knapsack, and a sorted list of unassigned items

```
for each item in the sorted list do
    if the size of this item is no greater than the bin's capacity then
        Assign this item to the bin.
        Remove this item from the unassigned list.
        Subtract the bin's capacity by the size of this item.
```

unassigned sorted item list as possible. Label packed items as "assigned", otherwise "unassigned".

ii) (Rearrangement): Clear all items from knapsacks but retain the "assigned" or "unassigned" information. Run a cyclic allocation in Algorithm 2 only for those assigned items. During this process, relabel these items if necessary. Finally, run the greedy step in Algorithm 1 again over all the knapsacks with the rest of the "unassigned" items.

iii) (Swap and fit): For each pair of assigned items, try to see if swapping them will create more space for more item to fit in.

iv) (Delete and fit): For each assigned item, try to see if removing this item in its knapsack would create space for more items to fit in.

The VMs are considered "packable" if and only if the maximized packed value equals the total value.

2) One simple counter example: We use the following simple counter example to show that for certain lists of items, *MTHM* will return "unpackable", even though the items are truly "packable" into the same knapsacks.

Consider 2 knapsacks with capacity 100, a list of items sized $[11, 12, 13, 14, 27, 44, 71]$. There is one packing solution: items sized $[11, 13, 71]$ in the first knapsack, items sized $[12, 14, 27, 44]$ in the second knapsack. But MTHM will not return this solution.

Let us see each step of *MTHM*.

1) Initialization step will label all the item except "71" as "assigned". The greedy result is: $[11, 12, 13, 14, 27]$ in the first knapsack, $[44]$ in the second knapsack.
2) In rearrangement step, the cyclic allocation process will have the following. The first knapsack has $[44, 14, 12]$, and the second knapsack has $[27, 13, 11]$. Then 71 can not fit into any of the two in the final greedy step.
3) In swap and fit step, the largest possible space increase for the first bin can be made by "swapping out 44, and swapping in 11". This only increases the space from 30 to 63, which is not enough for 71. Similarly, for the second knapsack, the largest possible space increase can come from "swapping out 27, and swapping in 12". Then it only increases the space from 49 to 64 which is still not enough for 71.
4) The delete and fit step simply does not contribute at all towards getting a packing solution.

Algorithm 2 Cyclic allocation in MTHM

Given a sorted list of assigned items

```
s ← 0
for each item in the sorted list of assigned item do
    Let l be the first knapsack index in {s, s+1, ..., m−1, 0, ..., s−1} such that knapsack l's capacity is no less than this item's size.
    if There is no such l then
        Label this item back to "Unassigned".
    else
        Assign this item to knapsack l.
        Subtract knapsack l's capacity by the size of this item.
        if l + 1 == m then
            s ← 0.
        else
            s ← l + 1.
```

B. Gold-dust world: when VM sizes are small

We study the performance of *Water-filling* when the maximum VM size s_{max} is much smaller than the minimum host capacity c_{min}. Particularly, we show that the number of items that *Water-filling* will allocate is close to the number achieved by OPT the optimal algorithm.

We show the following two results.

i) For packing, when $s_{max} \ll c_{min}$, the number VMs of *Water-filling* can pack is close to the number packed by OPT.

ii) Then for k-HA, when $s_{max} \ll c_{min}$, instead of keeping k standby hosts, we can just keep assigning VMs to hosts by *Water-filling*, until the total remaining space reaches some threshold. We state the corresponding claim and quantify the threshold in Theorem 5.

1) Bounds for packing when using Water-filling: Let there be m knapsacks with smallest of capacity c_{min} and let the largest of the items have size s_{max} with $s_{max} \ll c_{min}$. Let total capacity of all knapsacks be C_0, so $C_0 >= m \cdot c_{min}$.

Theorem 4: If N_{OPT} items can be packed into the bins by OPT then *Water-filling* will pack at least $(1 - s_{max}/c_{min}) * N_{OPT}$ items.

Proof: First, *Water-filling* will always pack at least $N_{waterfill} = N_{OPT} - m$ items. Consider the items left over by *Water-filling*, the smallest left over item is bigger the largest residual space else greedy would pack it. but we know the total volume of the residual space is greater than the total volume of the leftover items because OPT is able to pack everything. So the number of remaining items by greedy is at most m. (Otherwise, the total volume of the residual space must be less than the total volume of the leftover items.)

So $N_{waterfill} >= N_{OPT} - m$, i.e. $N_{worstfit}/N_{OPT} >= 1 - m/N_{OPT}$. So *Water-filling* achieves a ratio better than $1 - m/N_{OPT}$. Now $N_{OPT} >= C/s_{max}$. Therefore $1 - m/N_{OPT} >= 1 - m \cdot s_{max}/C >= 1 - m \cdot s_{max}/m \cdot c_{min} = 1 - s_{max}/c_{min}$. ■

2) Bounds for k-HA when using Water-filling: Now given that a set of VMs that are packable into m hosts, what HA level will such packing achieve? We answer this question by the following mechanism.

Let C_k denote the total capacity of all the hosts except the largest k hosts.

Theorem 5: k-HA is guaranteed, if k is the maximum number such that the inequality in Eq.(1) holds.

$$C_k - S > (m-k)(s_{max} - 1) \tag{1}$$

Proof: Given a k, $C_k - S$ denotes the residual capacity of all the hosts except the largest k hosts when all the VMs are packed into these $m-k$ hosts. Now, if $C_k - S > (m-k)(s_{max}-1)$, by *Pigeon Hole Principle*, then there is at least one host that has residual capacity that is strictly greater than $s_{max} - 1$ (at least s_{max}), i.e. one other VM can fit into the residual space. ■

Now, given a set of m hosts, and a sequence of incoming VMs, we can actually keep allocating VMs to hosts by *Water-filling*. At each assignment, we need to update all m inequalities, the number of such inequalities that hold gives the number of HA level.

C. Doubling world: VM sizes in the form of 2^i

We show that when the VM sizes are of the form: $2^i (i = 0, 1, 2, \dots)$, VMP can be solved exactly in polynomial time. In particular, *Water-filling* not only finds the feasible packing if it exists but it also achieves load balancing.

We first give the intuition for why packability is decidable in polynomial-time in the *doubling world*. We are given a list of n VMs in decreasing order of size, the VM sizes being of the form $2^i (i = 0, 1, 2, \dots)$, i.e. 256GB, 512GB, etc., and a fixed number of k hosts with arbitrary capacity (does not need to be of any special form). Consider the first VM in the list; one of the two following cases must happen.

i) There is no space on any host to accommodate this VM. Then clearly, this set of VMs is simply not able to be packed/repacked.
ii) There are multiple hosts that can accommodate this VM. Then, it does not matter which host we place this VM on, because the remaining VMs can always well-utilize the empty space which the first could have been put into.

Then we claim that given a set of k hosts *Water-filling* can exactly check the packability of the list of VMs, whose sizes are in the form of some power of 2.

We develop the following definitions and theorems to support the above argument.

Definition 1 (General packing): Any packing that just satisfies the capacity constraint is a general packing.

Definition 2 (Canonical packing): The packing solution returned by *Water-filling* is called *canonical packing*.

Definition 3 (Packing level of each VM): Let j denote the host to which VM i is assigned by *Water-filling*. Let l_{ij} denote the total size of VMs packed on host j before VM i is placed.

Canonical packing has the following necessary and sufficient condition:

Property 1: A *general packing* is a *canonical packing*, if and only if for any packed item on any host, there is no strictly smaller item on another host that has a strictly smaller packing level.

Theorem 6: There is a general packing, if and only if there is a canonical packing.

Proof: "⇐": this direction is trivial since any canonical packing is a general packing.

"⇒": Let VMs in each host be ordered from largest at the bottom to smallest at the top; Then across different hosts, for any two different hosts s, t, if a larger VM on host s is at a higher packing level than a smaller VM on host t, doing a swap still gets a valid packing. Do such swaps until no more swap can be done. We have a canonical packing. ■

Corollary 2: If *Water-filling* decides no canonical packing exists, then there is no general packing.

This guarantees no false negative when *Water-filling* returns a non-packable decision.

D. Performance evaluation

We evaluate *Water-filling* vs *MTHM* by our IID-IK model. Given host capacity and VM size distribution, repeat the process of random sampling from both of the two distributions described in Section VI for 500 times, we get the asymptotic metric $\mathbb{PE}$ in each setting.

1) A small dataset: First, we show how much worse *MTHM* performs while *Water-filling* achieves optimality. We compute $\mathbb{PE}$ for *Water-filling* and *MTHM* given the following host/VM size distribution in Table I.

TABLE I
ONE SMALL HOST/VM SIZE DISTRIBUTION

Host size (GB)	Frequency (%)	VM size (GB)	Frequency (%)
100	100	2	16.66
		4	16.66
		8	16.66
		16	16.66
		32	16.66
		64	16.66

TABLE II
ASYMPTOTIC METRIC $\mathbb{PE}$ FOR THE SMALL DISTRIBUTION

Number of hosts	$\mathbb{PE}$ for Water-filling	Ratio for MTHM
10	4.84	3.98
20	4.72	3.68
30	4.77	3.73
40	4.74	3.65
50	4.74	3.65
60	4.77	3.68
70	4.77	3.69
80	4.78	3.68
90	4.75	3.69
100	4.76	3.67

Table II shows *MTHM* performs $\sim 21\%$ worse than *Water-filling* which achieves optimality in this dataset with a small support for the distributions.

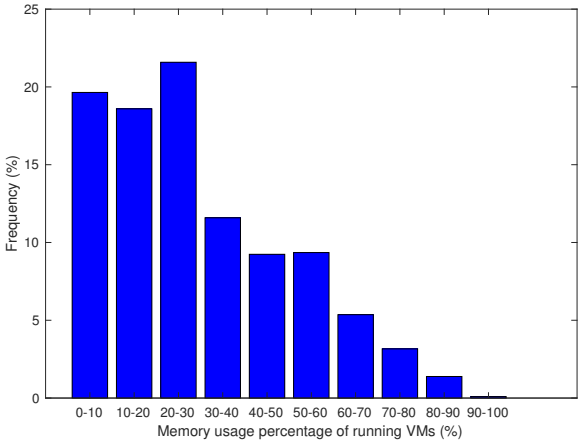

Fig. 1. Distribution of memory usage by running VMs

2) Nutanix workload & asymptotic metric $\mathbb{PE}$ results: In Nutanix's data center, host-nodes (hosts) are deployed inside clusters. Each cluster contains a number of hosts. VMs share physical resources on each host.

In this snapshot of a typical workload at Nutanix, there are 5499 clusters. 2687 of them have 4 host nodes, the remaining 2872 of them have 3 host nodes.

Fig. 1 shows the memory in percentage consumed by those running VMs on their host nodes. We can see the majority of memory consumption on those host nodes are under 40%.

Fig. 2 shows the host capacity distribution. Fig. 3 shows the VM size distribution.

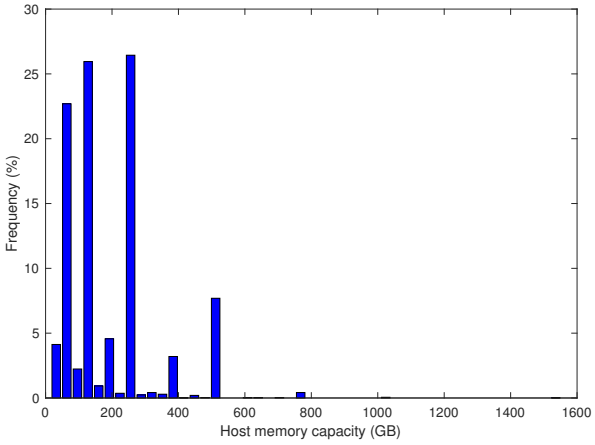

Fig. 2. Distribution of host memory capacity

Table III shows the asymptotic metric $\mathbb{PE}$ for host/VM size distribution from the unmodified Nutanix workload. As we can see from Fig. 2 and 3, the VM sizes are much smaller than the bin sizes. This causes the asymptotic metric $\mathbb{PE}$ of *Water-filling* to be slightly better than *MTHM*'s, which also supports our previous conclusion on *Gold-dust World* in Section VII-B.

We round VM sizes up to the closest 2^i GB, and merge the

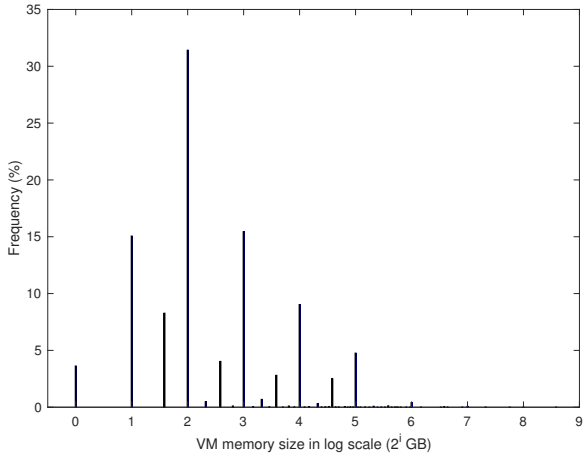

Fig. 3. Distribution of VM memory size

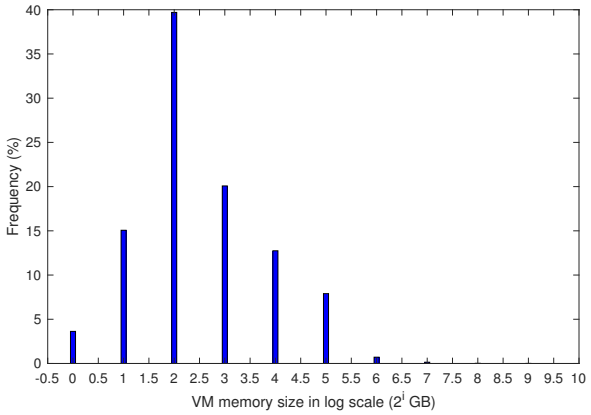

Fig. 4. Distribution of VM memory size after rounding

distribution entries with the same rounded-up sizes. We get the rounded-up memory size distribution for VMs in Fig. 4.

Table IV presents the asymptotic metric for this setting. *Water-filling* achieves optimality due to the conclusion on *Doubling World* in Section VII-C. *Water-filling* still performs slightly better than *MTHM* due to the large gap in size between hosts and VMs.

TABLE III
ASYMPTOTIC METRIC $\mathbb{PE}$ FOR ORIGINAL HOST/VM SIZE DISTRIBUTION FROM NUTANIX

Number of hosts	$\mathbb{PE}$ for *Water-filling*	Ratio for MTHM
10	23.253	22.905
20	23.996	23.732
30	23.136	22.898
40	23.101	22.855
50	23.798	23.591
60	23.670	23.453
70	23.668	23.462
80	23.484	23.275
90	23.265	23.060
100	23.666	23.454

TABLE IV
ASYMPTOTIC METRIC $\mathbb{PE}$ FOR POWER-OF-2

Number of hosts	$\mathbb{PE}$ for *Water-filling*	Ratio for MTHM
10	21.831	21.831
20	21.730	21.730
30	21.881	21.878
40	21.399	21.398
50	21.485	21.483
60	21.141	21.138
70	21.973	21.971
80	21.407	21.407
90	21.464	21.457
100	21.555	21.551

E. Water-filling packs best

To summarize, *Water-filling* has the following nice properties.

- *Water-filling* places VMs into the emptiest host at each step, which leads to a most balanced load distribution among hosts.
- Given sorted lists of VMs and hosts in order of size, *Water-filling* runs in $\theta(mn)$ time, while *MTHM* takes $\theta(mn+n^2)$. In industrial practice, the number of hosts is often a constant. Thus, *Water-filling* satisfies the stringent real-time requirements of commercial applications.
- *Water-filling* is proved to perform very close to the optimal in *Gold-dust World* where VM sizes are much smaller than host sizes.
- Also in *Gold-dust World*, we develop simple inequalities to check the level of HA when applying *Water-filling* in Theorem 5, which makes the computation of k-HA placement even faster.
- Simulation on industrial workload from Nutanix shows *Water-filling* has better asymptotic metric than *MTHM* in our IID-IK model.

In other words, *Water-filling* is the best.

VIII. CONCLUSION

In this paper, we study the HA problem in VM placement and prove novel complexity-theoretic results, particularly the surprising result that the existence of a k-HA placement is in NP. We leave open the problem of deciding the exact complexity of deciding whether a given placement is k-HA or not. We propose a natural stochastic model - IID-IK model - which provides a uniform basis for comparing heuristics. We also show that interestingly there exists a *best* heuristic in this model whose packing efficiency is computable. We utilize this model and analyze a variety of heuristics used in practice; we also consider natural special cases of input distributions that occur in the real world and conclude that *Water-filling* is the best.

REFERENCES

[1] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 700–709.

[2] S. Nanda and T. cker Chiueh, "A survey of virtualization technologies," StonyBrook University, Tech. Rep., 2005.

[3] I. Pietri and R. Sakellariou, "Mapping virtual machines onto physical machines in cloud computing: A survey," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 49:1–49:30, Oct. 2016.

[4] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. F. Chan, "Leveraging virtualization to optimize high-availability system configurations," *IBM Systems Journal*, vol. 47, no. 4, pp. 591–604, 2008.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[6] S. Martello and P. Toth, "Heuristic algorithms for the multiple knapsack problem," *Computing*, vol. 27, no. 2, pp. 93–112, 1981.

[7] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications," *IEEE Transactions on dependable and Secure Computing*, vol. 10, no. 5, pp. 273–286, 2013.

[8] S. Yang, P. Wieder, and R. Yahyapour, "Reliable virtual machine placement in distributed clouds," in *Resilient Networks Design and Modeling (RNDM), 2016 8th International Workshop on*. IEEE, 2016, pp. 267–273.

[9] M. Jammal, A. Kanso, and A. Shami, "Chase: Component high availability-aware scheduler in cloud computing environment," in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 477–484.

[10] A. Jahanbanifar, F. Khendek, and M. Toeroe, "Providing hardware redundancy for highly available services in virtualized environments," in *Software Security and Reliability, 2014 Eighth International Conference on*. IEEE, 2014, pp. 40–47.

[11] S. Shen, A. Iosup, A. Israel, W. Cirne, D. Raz, and D. Epema, "An availability-on-demand mechanism for datacenters," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 495–504.

[12] A. Zhou, S. Wang, Z. Zheng, C.-H. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Transactions on Cloud Computing*, vol. 4, no. 4, pp. 452–466, 2016.

[13] K. Jansen, "A fast approximation scheme for the multiple knapsack problem," in *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 2012, pp. 313–324.

[14] E. G. Coffman, K. So, M. Hofri, and A. Yao, "A stochastic model of bin-packing," *Information and Control*, vol. 44, no. 2, pp. 105–115, 1980.

[15] P. W. Shor, "the average-case analysis of some on-line algorithms for bin packing," *Combinatorica*, vol. 6, no. 2, pp. 179–200, 1986.

[16] E. G. C. Jr., C. Courcoubetis, M. R. Garey, D. S. Johnson, L. A. McGeoch, P. W. Shor, R. R. Weber, and M. Yannakakis, "Fundamental discrepancies between average-case analyses under discrete and continuous distributions: A bin packing case study," in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, 1991, pp. 230–240.

[17] V. Gupta and A. Radovanovic, "Online stochastic bin packing," *CoRR*, vol. abs/1211.2687, 2012. [Online]. Available: http://arxiv.org/abs/1211.2687

[18] C. M. Papadimitriou, *Computational complexity*. Reading, Massachusetts: Addison-Wesley, 1994.

[19] I. Smeets, A. K. Lenstra, H. Lenstra, L. Lovász, and P. van Emde Boas, "The history of the lll-algorithm," in *The LLL Algorithm - Survey and Applications*, 2010, pp. 1–17.

[20] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 702–710.