Compute Host Utilization – Implementing the Perfect Scheduler

Amrith Kumar, Fellow, Verizon [amrith.kumar@verizon.com]

Abstract

We examine the placement of virtual machines in an OpenStack deployment, and explore possible algorithms to optimize the utilization of the infrastructure.

OpenStack Instance Placement

In OpenStack, a *compute host* is a server on which virtual machines are scheduled. In response to a user-request, the Nova service invokes a scheduler that looks across a fleet of compute hosts and determines where to place the requested virtual machine. *Affinity* and *anti-affinity* policies further allow for the specification of placement of virtual machines with respect to other virtual machines. Scheduling is based on available resources on the compute host. OpenStack tracks and schedules VMs based on vCPU, memory, and local disk requirements. Over-subscription is a mechanism whereby a single instance of a resource on a compute host could be allocated to multiple virtual machines. Over-subscription of a resource (vCPU, memory, and disk) are defined on for each compute host.

Requests are not known ahead of time. The order in which requests are received is not deterministic. This is because requests are received when an application is brought onto the system. The OpenStack cluster is a multi-tenant system with a number of applications coexisting on a set of shared compute hosts. In the most general case of the problem, an OpenStack cluster has many compute hosts, and many applications which come on to the platform over a period of time.

The Perfect Scheduler

We now define a useful construct called the "perfect scheduler".

Consider the set of applications which are all going to be (over time) brought onto an OpenStack cluster. Assume that it known in advance what virtual machines each of them will be requesting, what resources are required for each of them, what affinity and anti-affinity rules will be specified, and what oversubscription is being used on each compute host, for each resource. Assume further that there are a set of possible virtual machine placements across the compute hosts that will allow all requests to be satisfied with the minimum number of compute hosts used. Let us call this set $\{\boldsymbol{S}\}$.

The "perfect scheduler" is an algorithm that will receive and process requests for placement of virtual machines with no knowledge of what requests will be received in the future. The Perfect Scheduler places virtual machines on the compute hosts in such a way that when all requests have been received and processed, the state of the cluster $\boldsymbol{s}$ is in $\boldsymbol{S}$ (i.e. $\boldsymbol{s} \varepsilon \boldsymbol{S}$).

Similarity to known problems

This problem is similar to the traditional "bin packing" problem which attempts to pack a

number of fixed sized objects into a collection of bins of fixed (and identical) size so as to occupy the least volume in total. The solution to this problem is known to be NP-Hard[1].

Another similar class of problems is the "0/1 knapsack problem" which attempts to maximize the weight of objects of different weights placed into a knapsack, subject to some upper bound. This problem is also known to be NP-Hard[2].

This problem appears to resemble the issue of stacking blocks in the game of Tetris. The algorithms for packing in Tetris are well understood and the greedy (lamebrain) algorithm is known to work quite well.

However, the problem at hand is not like any of these problems in some important respects.

First, each placement request must be satisfied with incomplete information – what requests will follow, and the order in which they will follow is not known. In this regard, the problem at hand is like the Tetris problem.

Second, in the traditional bin-packing problem in 3-space, every unit of space in the bin may be used. In the VM packing problem this is not the case. Consider a vCPU that is defined to be subscribed at a ratio of N:1. That means that at most N virtual machines may be allocated to each vCPU. Without loss of generality, we can reduce any N:1 oversubscription into a 1:1 subscription model by merely assuming that the compute host has N vCPUs for each vCPU actually present in the compute host. In other words, a compute host with 32 virtual cores and subscribed as 3:1 can be modeled as a compute host with 96 vCPUs where each vCPU can only be assigned to a single virtual machine.

Third, in Tetris, when the lowest row is solid, it gets consumed and the whole board moves down. There is no equivalent in the virtual machine packing problem.

Simplified graphical representation

As described in the preceding section, the virtual machine placement problem is not the traditional bin-packing problem, but a similar graphical representation can be created.

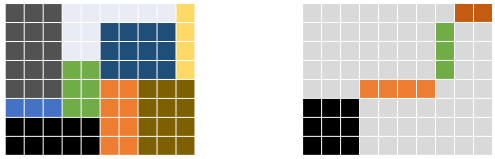

Figure 1. The traditional bin-packing problem (at left) and the Virtual Machine packing problem (at right).

On the left, the traditional bin-packing problem in 2-space. We show a total of 8 objects packed into a grid of 10x8. On the right, the VM packing problem showing four VMs occupying the same 10x8 grid. In this representation, we have reduced all resources to their 1:1 subscribed equivalent. Without loss of generality this representation can be extended into N-space.

[1] Determining whether or not a solution exists is known to be NP-complete.
[2] Determining whether or not a solution exists is known to be NP-complete.

Computing resource utilization in N-space

We model a system consisting of a set of boxes that must be filled with objects of varying sizes. In the most general N-space version of this problems, boxes are N-dimensional containers into which N-dimensional objects are placed.

In a simple 2-space representation we now illustrate the available system resources, and show how we compute utilization.

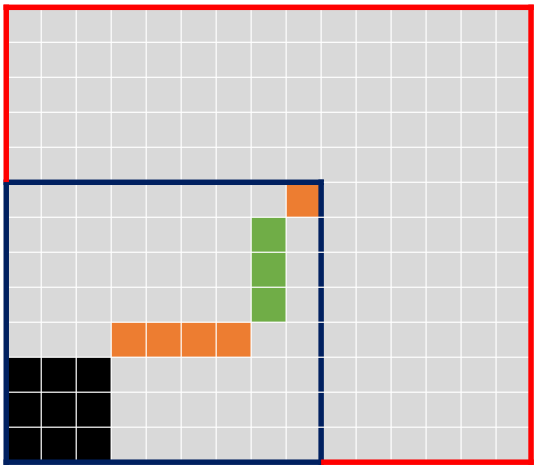

Figure 2. Showing a 2-space system, the available system capacity, and the capacity utilized.

In this 2-space system, the red box shows the total system resources available, and the smaller blue box that bounds the four objects represents the total system resources consumed.

Assume an N-space box of dimensions [B_1, B_2, … B_N] inside which there are M objects of dimensions [$D_{1,1}$, $D_{1,2}$, … $D_{1,N}$], [$D_{2,1}$, $D_{2,2}$, … $D_{2,N}$] … [$D_{M,1}$, $D_{M,2}$, … $D_{M,N}$].

The total resources available in this system is $\prod_{i=1}^{N} B_i$, and the total resources consumed in this system is $\prod_{i=1}^{N}\left(\sum_{j=1}^{M} D_{i,j}\right)$.

The resource utilization is therefore computed to be $\frac{\prod_{i=1}^{N}\left(\sum_{j=1}^{M} D_{i,j}\right)}{\prod_{i=1}^{N} B_i}$.

It is worth observing that in the standard bin packing problem, one would compute the bin space utilization as $\frac{\sum_{i=1}^{N}\left(\prod_{j=1}^{M} D_{i,j}\right)}{\prod_{i=1}^{N} B_i}$ and these two are not equal.

Problem 1

A system consists of a set of boxes of equal size, that must be filled with objects of random sizes. Upon receipt of each object, the system must first place it in a box before moving on to the next object.

What strategy should be used to pick a box in which to place the object?

When placing an object, one strategy is to find the emptiest box that will accommodate the object, and place the object there. In effect, this will attempt to evenly distribute objects across all boxes.

A second strategy is to find the fullest box that will accommodate the object, and place the object there. This strategy will attempt to pack boxes as much as possible, before attempting to use an empty box.

A third strategy is to place the object in some random box that will accommodate the object.

We compare the performance of these three strategies by simulating a system with a number of boxes, and generating pseudo-random streams of objects.

In this simulation, various numbers of boxes, various box sizes, and various ranges of possible object sizes were attempted.

We present the results of an exemplar simulation below. Assume a system with 50 boxes of size 100, and objects of random size between 1 and 30. A pseudo-random stream of objects is generated such that the sum of the size of all objects is equal to the total capacity of the system (50 * 100 = 5000). With this stream of objects, the system attempts to place them sequentially till we reach a situation where an object has no viable box. The three placement strategies described above were simulated.

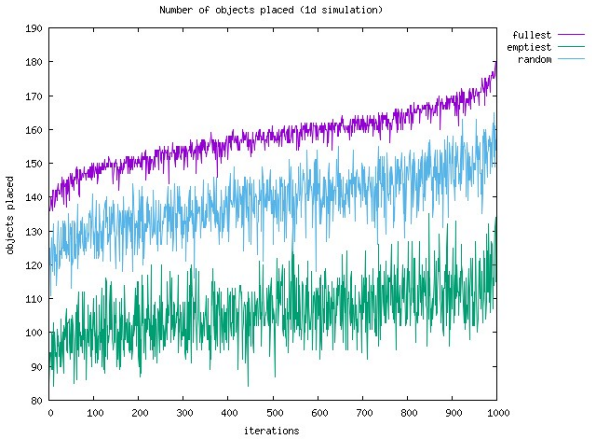

Figure 3. Shows the number of objects successfully placed using three strategies

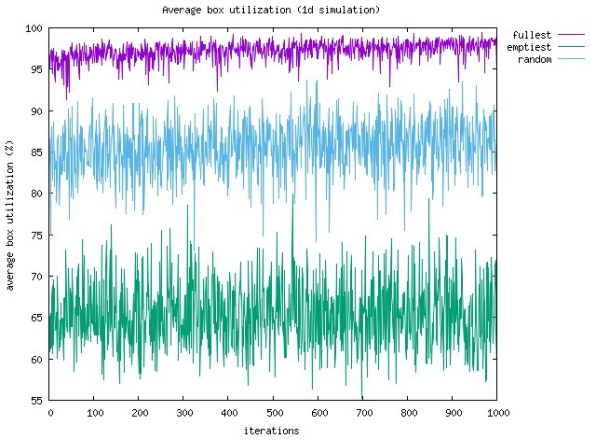

Figure 4. Showing the average utilization of boxes at the end of the simulation.

We observe that the placement strategy that choose the emptiest box which would accommodate an object has the worst overall performance, and the strategy that chooses the fullest box which would accommodate an object has the best overall performance.

Problem 2

The earlier problem described the placement of one-dimensional objects. In practice, we wish to understand the best strategies for placement of multi-dimensional objects.

A system consists of N-dimensional boxes of varying sizes. N-dimensional objects of varying sizes must be placed into these boxes but the sum of the *i*-th dimension of all objects in a box may not exceed the *i*-th dimension of the box.

We adopt a similar simulation strategy and test the same three placement strategies. In determining how full a box is, we compute the bounding n-dimensional polygon that encompasses all objects that are in a box. In the exemplar simulation below, we used two-

dimensional objects, and boxes with two-dimensional limits. The results are shown below.

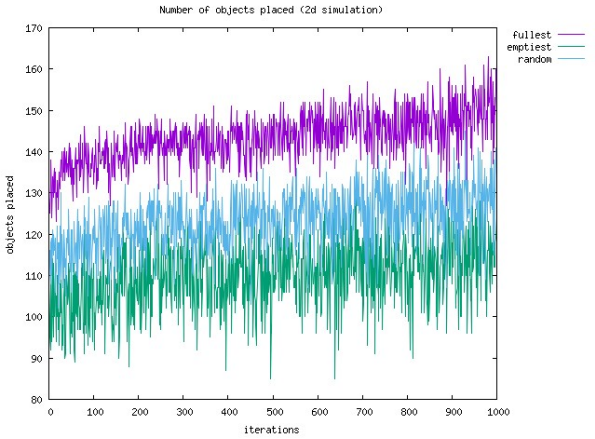

Figure 5. Shows the number of objects successfully placed using three strategies with two-dimensional objects

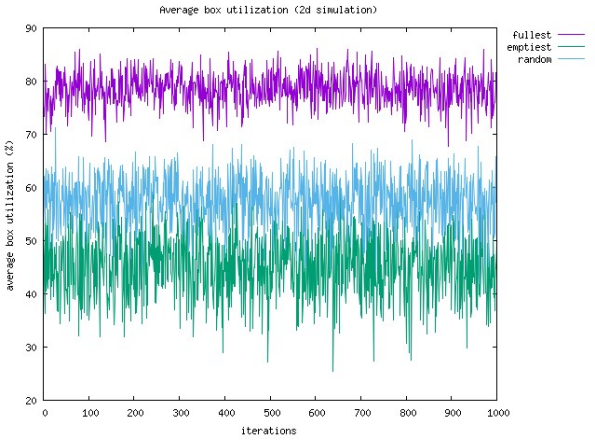

Figure 6. Showing the average utilization of boxes at the end of the simulation with two-dimensional objects.

We observe again that a placement strategy based on fullest box provides the best overall performance and we achieve utilization in excess of 2x over a strategy that places objects in the emptiest box.

Problem 3

In the simulation above, the limit on the sizes of boxes was the same in all dimensions, and the minimum and maximum sizes of objects were also the same on all dimensions.

In practice, boxes need not of equal size on all dimensions. Furthermore, there may be boxes of different sizes in the system. The simulation was repeated with different sizes (and object size limits) on the different dimensions, and boxes of different sizes. The results of those simulations are presented below.

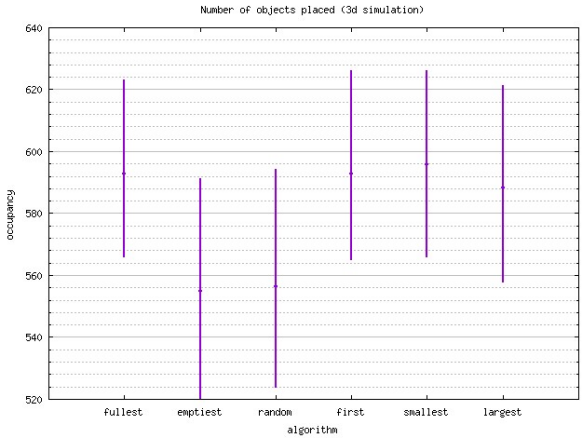

Figure 7. Shows the number of objects successfully placed using three strategies with three-dimensional objects and varying sized boxes with different sizes on each dimension.

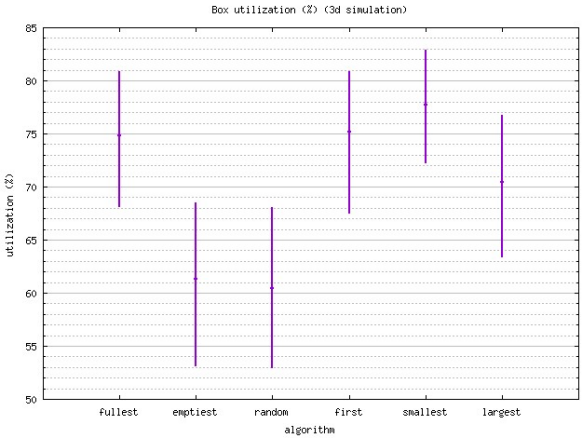

Figure 8. Showing the average utilization of boxes at the end of the simulation with three-dimensional objects and varying sized boxes with different sizes on each dimension.

Since boxes have different sizes, we have some additional choices in object placement. We simulated six possible placement strategies, “fullest”, “emptiest”, and “random” as before, and “first” which places the object into the first box that would accommodate the object, “smallest” which places the object into the smallest box that would accommodate the object, and “largest” which places the object into the largest box that would accommodate the object.

The graphs in Figure 7, and Figure 8 show the minimum, maximum, and average count and utilization in each of these strategies.

Shown below in Figure 9, and Figure 10 are similar results for simulation in two dimensions, and

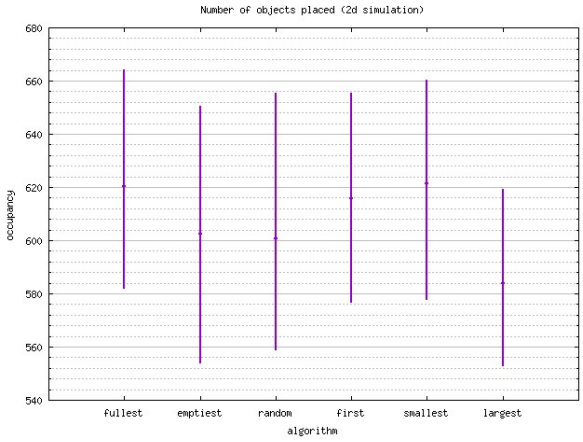

Figure 9. Shows the number of objects successfully placed using three strategies with two-dimensional objects and varying sized boxes with different sizes on each dimension.

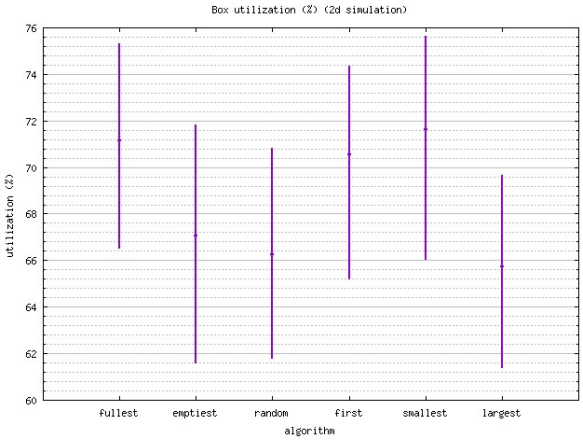

Figure 10. Showing the average utilization of boxes at the end of the simulation with two-dimensional objects and varying sized boxes with different sizes on each dimension.

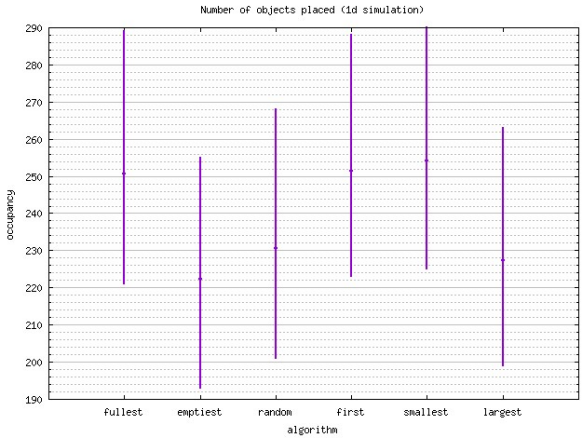

Figure 11. Shows the number of objects successfully placed using three strategies with one-dimensional objects and varying sized boxes with different sizes on each dimension.

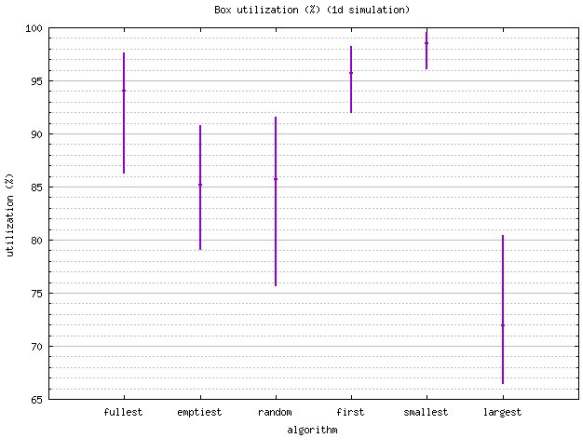

Figure 12. Showing the average utilization of boxes at the end of the simulation with one-dimensional objects and varying sized boxes with different sizes on each dimension.

Observations

The simulations presented above provide a clear indication that packing boxes is the best strategy to achieve the best overall utilization, and the best strategy to maximize the number of objects that can be placed in boxes in situations where boxes are all of identical size. When boxes of different sizes are available to the system, placing objects into the fullest box allows for the placement of the most objects, but placement into the smallest box allows for the best overall system utilization. However, in all cases, placing objects into the emptiest box is the worst strategy all around.

Further Analysis

The simulations above only deal with placement of objects into a single set of boxes. Relating this to the placement of virtual machines in OpenStack, this is equivalent to placement of virtual machines on the default host aggregate, and with no affinity or anti-affinity rules.

While it is intuitive that the fullest strategy, or the smallest strategy will provide the best results even with multiple host aggregates, affinity, and anti-affinity rules, it may be worthwhile to run additional simulations and verify this to be the case.