Performance Best Practices for Kubernetes with VMware Tanzu

Performance Study - March 15, 2021

VMware Tanzu™

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

Table of Contents

Executive Summary

VMware Performance Engineering tested the limits of VMware Tanzu with Kubernetes performance in a variety of situations using two Tanzu editions: Basic and Standard. Basic uses VMware vSphere, while Standard uses the VMware Cloud Foundation platform. The team collected best practices during testing to help users get the best performance from their VMware Tanzu with Kubernetes clusters.

Introduction

This technical paper gives performance best practices for running Kubernetes with two types of deployments:

- VMware vSphere® 7 with VMware Tanzu™ – also known as VMware Tanzu Basic Edition (https://tanzu.vmware.com/tanzu/basic)
- VMware Cloud Foundation™ (based on vSphere 7) with VMware Tanzu™ – also known as VMware Tanzu Standard Edition (https://tanzu.vmware.com/tanzu/standard)

These platforms provide integrated solutions for deploying developer and production-ready clusters for Kubernetes workloads.

This document guides administrators and developers seeking to maximize workload performance and resource efficiency when configuring Kubernetes clusters and deploying workloads on these solutions. This includes guidance on choices such as:

- Sizing VMware Tanzu™ Kubernetes Grid™ (TKG) clusters and selecting virtual machine classes for worker nodes
- Configuring pod and container resource specifications
- Choosing the right level of CPU and memory over-commitment with the least impact on workload performance

Intended audience

vSphere administrators, Kubernetes cluster administrators, DevOps engineers, and developers

Related documents

This document assumes you are familiar with the concepts underlying vSphere with Tanzu and vSphere Cloud Foundation with Tanzu. These concepts are discussed in the product documentation for vSphere with Tanzu at https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-152BE7D2-E227-4DAA-B527-557B564D9718.html.

You should also follow the general performance best practices for vSphere 7, as documented in the guide *Performance Best Practices for VMware vSphere 7.0* at https://www.vmware.com/techpapers/2020/vsphere-esxi-vcenter-server-70-performance-best-practices.html. This document contains performance recommendations specific to vSphere with Tanzu.

Methodology

The best practices discussed in this paper were developed and validated through extensive testing with open-source performance benchmarks. The benchmarks were run with multiple values for tuning and configuration options. The options that provided the greatest improvement in performance across multiple benchmark runs were identified as best practices.

Most of the example data provided in this paper was generated using the Weathervane 2.0 performance benchmark (https://blogs.vmware.com/performance/2020/02/weathervane2-kubernetes.html). Weathervane 2.0 is an open-source benchmark that compares the performance characteristics of on-premises and cloud-based Kubernetes clusters. Weathervane is an application-level benchmark, which means that it includes a representative application that is deployed on the clusters under test. The impact of the configuration or tuning of the clusters on the application's performance can be compared to determine which cluster provides the best performance for similar applications.

The application provided by Weathervane is a multi-tier web application which includes stateless web and application services, as well as stateful data services. You can deploy the application in multiple pre-tuned configurations which vary the number of replicas of each service and the amount of data used in each run. You can use multiple instances of the application in a single run of the benchmark to scale up the load on the clusters under test.

Weathervane includes a workload driver, which drives load on the clusters by simulating users interacting with the application instances. The performance metric for a run of Weathervane is the maximum number of simulated users that can interact with the application instances without violating any of the predefined quality-of-service metrics. This metric is called WvUsers, and a higher result indicates better performance. You can find more information about Weathervane 2.0 at https://github.com/vmware/weathervane.

Additional tests were performed using the Redis Memtier benchmark that is included in K-Bench, which is a framework to benchmark the control and data plane aspects of a Kubernetes infrastructure. Additional information about K-Bench can be found at https://github.com/vmware-tanzu/k-bench.

Best Practices for Individual Tanzu Kubernetes Grid Clusters

This section describes concepts and best practices that are relevant to the creation of individual clusters with the Tanzu Kubernetes Grid service (TKG clusters). If you will be deploying multiple TKG clusters on your infrastructure, you should also understand the best practices from the section Best Practices for Multiple Tanzu Kubernetes Grid Clusters before provisioning TKG clusters.

Select a virtual machine size for worker nodes

Problem description

When provisioning a cluster with the Tanzu Kubernetes Grid service, you must select a virtual machine class for the worker nodes. The virtual machine class determines both the resources allocated to the nodes and whether those resources are fully reserved. The virtual machine class types available for a TKG cluster are discussed in the product documentation at https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-tanzu/GUID-7351EEFF-4EF0-468F-A19B-6CEA40983D3D.html. This section discusses best practices related to selecting the size of the worker nodes. The next section discusses best practices related to the quality-of-service aspects of virtual machine class types.

The size selection of a virtual machine class controls the number of CPUs and amount of memory allocated to the virtual machine that hosts a worker node for the TKG cluster. These range from **xsmall**, which allocates 2 CPUs and 2GB of memory, to **8xlarge**, which allocates 32 CPUs and 128GB of memory. The resources allocated to a worker node cannot be changed once the node is deployed. However, it is possible to change the cluster to use a different virtual machine class with rolling updates.

Best practices

Understand resource usage of pods

To select the appropriate size for the worker nodes, you must understand the expected resource usage of the Kubernetes pods to be deployed on the nodes.

The most common resources to specify for containers deployed in a Kubernetes pod are *CPU* and *memory*. CPU resources are typically specified in millicores, where 1000 millicores (1000m) represents a single CPU. Memory resources can be specified in terms of kibibytes (1 KiB == 1024 bytes), mebibytes (1 MiB == 1024^2 bytes), or gibibytes (1 GiB == 1024^3 bytes). Kubernetes uses these resource categories for the specification of resource *requests* and *limits*.

- **Requests** represent the lower-bound resource usage of a container. The CPU and memory requests for a pod are the total of the requests for all containers in that pod. Requests are used for the following purposes:
  - Tracking node resource availability: When a pod is scheduled on a node, that node's available resources are decremented by the requests associated with the pod.
  - Admission control: The Kubernetes scheduler will only schedule a pod on nodes with enough available resources to satisfy the pod's requests.
  - Pod placement: The Kubernetes scheduler uses a pod's requests, along with information about the available resources on the nodes, to select the best node on which to run the pod. The ranking that the scheduler uses to select the best node is controlled by the configured scheduler priorities.
  - Pod eviction: The combination of the request and limit specifications for a pod's containers are used to place that pod into a quality of service (QoS) class. When a node runs low on memory, pods with a lower QoS class are evicted before higher QoS pods. See the Kubernetes documentation for more information about QoS classes at https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/ and pod eviction at https://kubernetes.io/docs/tasks/configure-pod-container/assign-memory-resource/.
  - CPU Shares: The CPU request for a pod is used to configure the value of CPU shares for the pod's cgroup in the worker node's Linux OS. When there is high contention for a node's CPUs, pods receive a percentage of the available CPU time in proportion to the size of their shares.
- **Limits** represent a cap on the amount of a resource that can be used by a pod. For clusters created with the TKG service, limits are implemented by configuring the cgroup quota and period parameters in the Linux OS of the worker node. Note that while resource limits can be useful to enforce proper resource usage, they can have an adverse effect on workload performance. See *https://k8s.af/* for some articles describing possible pitfalls with Kubernetes limits.
  - Limits are implemented differently for vSphere pods created in the Supervisor cluster. See the section titled Best Practices for Deploying Workloads in vSphere Pods for a discussion of using limits with vSphere pods.

The requests and limits specified for pods are used to place each pod into a Kubernetes quality-of-service (QoS) class. There are three possible QoS classes for pods:

- **Guaranteed:** A pod is placed in the Guaranteed QoS class if it has memory and CPU requests and limits set for all its containers, and the value for each container's requests and limits is equal. Guaranteed pods are chosen last for eviction when a node runs low on memory and can take advantage of advanced CPU management policies. Critical workloads should typically run in Guaranteed pods.

> **NOTE:** It is important to distinguish between Guaranteed pods and TKG nodes with the Guaranteed QoS type. The Guaranteed pod QoS type affects the placement and eviction priority of a single pod in a Kubernetes cluster. The TKG node Guaranteed QoS type affects CPU and memory reservations at the vSphere level, and will provide a performance benefit to all pods running on that TKG node. The effect of choosing between the TKG node Guaranteed and BestEffort QoS types is discussed in the section Choosing between BestEffort and Guaranteed VM Classes.

- **Burstable:** A pod is placed in the Burstable class if at least one pod has CPU or memory requests set, but the pod does not meet the requirements of the guaranteed QoS class. Burstable pods will be evicted before guaranteed pods, but after BestEffort pods.
- **BestEffort:** A pod is placed in the BestEffort class if none of its containers specify resource requests. Because they do not specify resource requests, pods with the BestEffort class can be scheduled on Kubernetes nodes whose capacity has been allocated to the requests of higher priority pods. These pods can then use resources left idle by the higher priority pods. However, BestEffort pods have the lowest priority for CPU time, and are evicted first if a node runs low on memory.

The rules for pod eviction are more complicated when multiple pods of the same QoS class are running on a node that is experiencing memory pressure. Refer to the Kubernetes documentation for more details.

It is important to understand the expected resource usage of your pods, and to reflect that usage in the resource requests for the containers of those pods. Failure to specify accurate resource requests may have the following consequences:

- Specifying requests larger than the actual needs of a pod can lead to idle resources that are not available to other pods. This happens when the requests of pods assigned to a node consume most of the available resources, but the pods do not actually use those resources. The pods' request values are subtracted from the available resources on the node, limiting the set of nodes on which other pods can be scheduled.
- Specifying requests that are too low can cause the Kubernetes scheduler to make poor placement decisions for pods. When provided with inaccurate request values, the scheduler may place multiple resource-intensive pods on a node with insufficient resources. This leads to poor performance, or even to pod evictions in the case of high memory usage.

There are additional factors that should be considered when selecting the resource requests for your pods:

- Some workloads have resource demands that are bursty or that vary over time. This forces a choice between selecting resource requests that represent steady-state or peak demands. Selecting requests for steady state will allow more pods to be deployed on each worker node but may not leave enough resources for periods of high load. Selecting requests for peak load may leave idle resources at non-peak periods. There is no correct answer for every workload. Instead, you must consider factors such as the criticality of the workload and the characteristics of the other workloads that are sharing the cluster.

The value selected for the resource requests has the following implications for the selection of the virtual machine size for your worker nodes:

- You must provision nodes with sufficient resources to run the pods with the largest request values.
  - Example: If your largest pod requires 1500m CPU and 9GiB, then you must provision nodes with at least 2 CPUs and more than 9GiB of memory. For clusters created with the TKG service, this implies nodes of size *large* or larger.
- You should consider how well your pods will pack into the selected node size. It is possible to provision nodes that have, in total, sufficient resources for your workload's requests, but on which it is not possible to schedule all of the pods.
  - Example: Consider a workload which has three pods requesting 850m CPUs and three pods requesting 400m CPUs. These pods are requesting a total of 3750m CPUs. While it would seem that this workload could run in two BestEffort small worker nodes, each with 2000m CPUs, it is not possible to schedule the pods onto those nodes. Consider the following assignment of pods to nodes: Node1: 850m + 850m = 1700m, Node2: 850m + 400m + 400m = 1650m. There is still 650m CPU available between the two nodes, but neither has enough available to satisfy the requests for the final pod with a 400m request.

Understanding the resource requirements of your workload is not a trivial task. Determining these characteristics may require load testing, real-time resource usage monitoring, and an understanding of the expected steady-state and peak loads to be placed on the workload. However, accurate resource estimates will benefit both workload performance and overall infrastructure utilization.

Understand overhead of system pods

Each worker node runs system pods in addition to the pods deployed by user workloads. The resource requests for the system pods reserve some of the resources on each worker node. Those resources are not available for workload pods.

You can view the resources configured and available on each node by running the command `kubectl describe nodes` while logged into the context for your TKG cluster. For example, the image on the next page shows the output from the `kubectl describe node` command for a worker node using the BestEffort medium VM class. The output has been edited to remove unnecessary details. A worker node using the BestEffort medium VM class type has a total of 2 CPUs (2000m) and 8GiB configured. These values correspond to the values for CPU and memory in the `Capacity` section of the output. The amount of node resource that can actually be allocated to pods is shown in the `Allocatable` section. With respect to memory, 100MiB of the node's capacity is consumed as overhead and is non-allocatable. All of the CPU capacity is allocatable. In the `Allocated Resources` section of the output, you can see that 250 millicores are allocated to the CPU requests of the `calico-node-ftj9j` system pod. This reduces the CPU capacity available to workload pods to 1750m.

> NOTE: Figure 1 is on the next page.

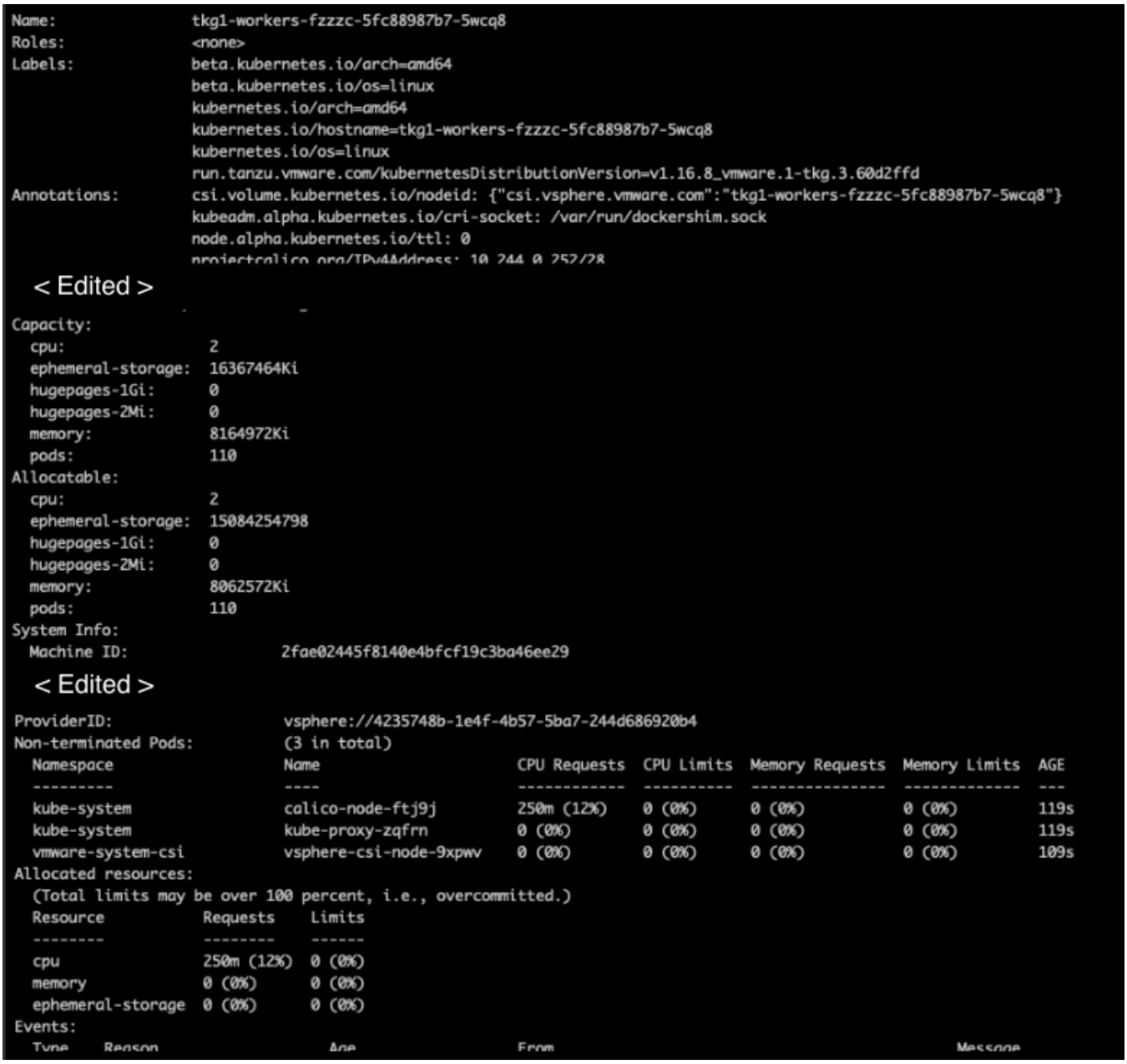

Figure 1. Node description provided by `kubectl describe node`

Choose TKG node sizes that fit into NUMA nodes

Modern multi-core servers use a memory architecture known as non-uniform memory access (NUMA). In a NUMA architecture, the entire memory space of the server is divided among groups of CPU cores. Each group, known as a NUMA node, has local memory which can be accessed with low latency. Processors in one NUMA node can access memory in a remote node, but at the cost of higher access latency. The VMware vSphere scheduler has optimizations to improve VM performance by minimizing the number of accesses to remote NUMA nodes. These optimizations work best when the VMs running on an ESXi host are sized to fit within a single NUMA node. This means that the number of CPUs and the amount of memory selected for a VM or TKG node should be smaller than the resources in a single NUMA node. The only exception is when you have individual pods whose resource needs are known to be larger than a NUMA node. Even in this case, you should consider whether that pod can be scaled out to multiple smaller pods.

You should refer to the product documentation for your server and processor model to determine the size of a NUMA node on your ESXi hosts. On some servers, each CPU socket corresponds to a NUMA node, while others have multiple NUMA nodes per socket.

Select the number of worker nodes

Problem description

When deploying a cluster using the TKG service, you must specify the number of worker nodes to be provisioned for the cluster. In combination with the VM class of the worker nodes, this determines the total amount of CPU and memory resources available to workload pods.

Note that you can scale up or scale down the number of worker nodes in a cluster created by the TKG service if you initially over-allocate or under-allocate nodes.

Best practices

Understand the total resource utilization of workloads

The importance of understanding the resource utilization of your pods was previously discussed in the context of selecting a worker node size. Understanding the total resource requirements of all pods to be deployed in a cluster is also important in the context of selecting the number of worker nodes.

To choose the minimum number of worker nodes needed to run your workloads, you must consider:

- The CPU and memory resources available on the selected node size after subtracting the overhead of system pods.
- The number of resources that will be unused after packing your workloads into the selected node size. These are the resources that are still available on the nodes, but that are too small to satisfy the requests of your smallest pods.

- Pods that have anti-affinity rules set to other pods and therefore will need to be placed on separate nodes. The minimum number of nodes needed will be the number that can satisfy the resource requests of all your pods after these overheads.

In addition to determining the minimum number of nodes, you should also consider the following factors when selecting the number of nodes:

- Peak resource usage: If the resource requests for your pods represent typical usage, but your workload may experience bursts of higher load, you should provision a sufficient number of nodes so that additional resources are available at peak load. Failure to leave resources available for bursty workloads may lead to poor performance or pods being evicted due to lack of resources.
- Auto-scaling: If you are using pod auto-scaling with your workload, you will need to provision additional worker nodes to accommodate anticipated peak load.

Understand overhead of system pods

As mentioned above, each worker node runs system pods in addition to the pods deployed by user workloads. These system pods reserve some of the resources on each worker node. Having a smaller number of larger worker nodes minimizes this overhead.

Demonstration: best practices in action

Understand workload resource requirements and available node capacity

As an example of how resource requirements and selecting the correct number of worker nodes impacts performance, consider the following scenario. We compared two possible clusters on which to run the applications of a business unit. The applications were represented by two instances of the small2 configuration of the Weathervane benchmark application.

With no knowledge of the resource requirements of the workloads, we initially configured the cluster depicted in figure 2. This cluster used twelve worker nodes which used the TKG best-effort xlarge VM class, which had 4 CPUs and 32GB. Note that in this case best-effort refers to the absence of reservations on the VM configured for the node, and it should not be confused with the BestEffort QoS class for Kubernetes pods. This cluster had a total of 48 CPUs and 384GiB.

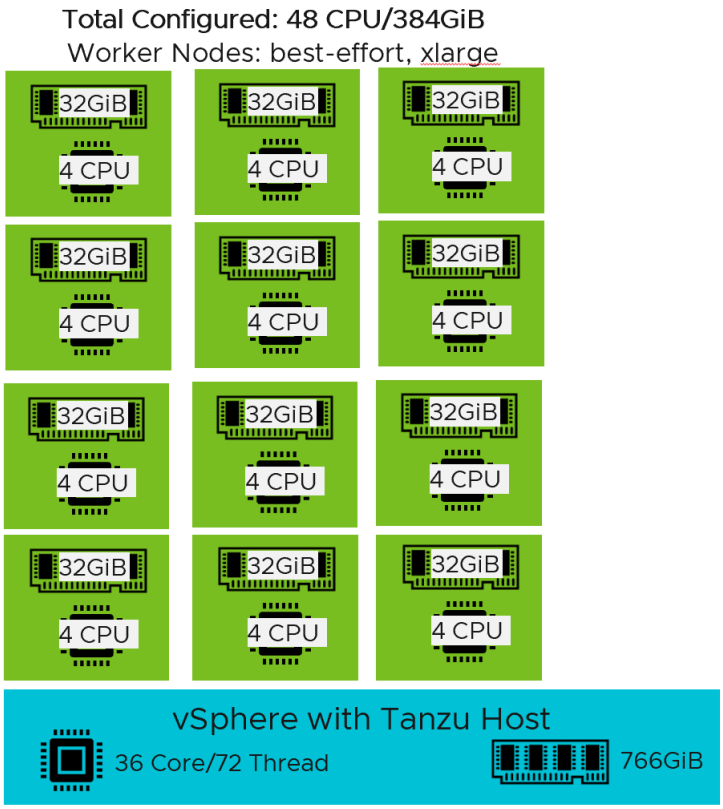

Figure 2. Cluster configured without knowledge of resource requirements

Using this cluster, the applications supported a total of 16,100 WvUsers, as shown in the chart in figure 5.

We then evaluated the resource usage of the applications. Each of the application instances in this experiment used the Weathervane small2 application configuration, which is shown in figure 3. In this figure, the pods are labelled with resource request values determined by monitoring the actual usage while under peak load. Based on these values, the workload only required a total of 9.6 CPUs and 64GiB, which meant that these applications could have been deployed on three best-effort xlarge nodes, which would have a total of 11.25 CPUs and 95GiB available after the overhead for system pods. However, the resource evaluation also showed that it was necessary to leave additional capacity on the nodes to allow for occasional bursts in demand. This led to the configuration of four best-effort xlarge worker nodes shown in figure 4.

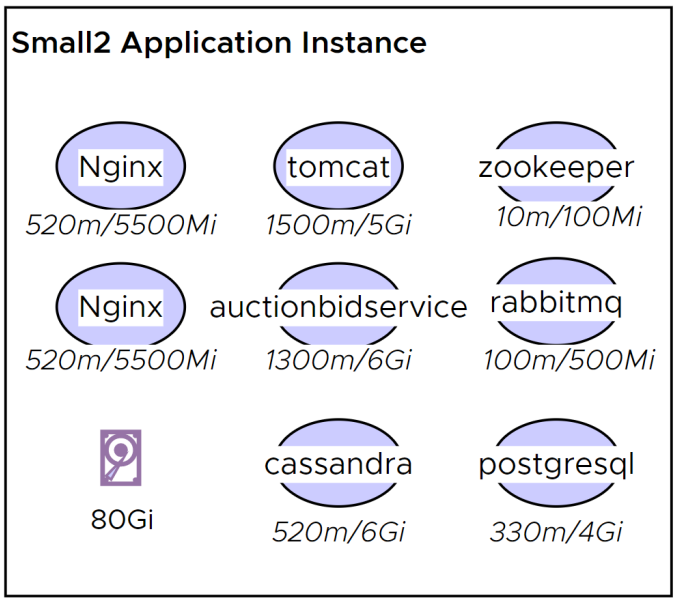

Figure 3. Weathervane small2 configuration with resource requests

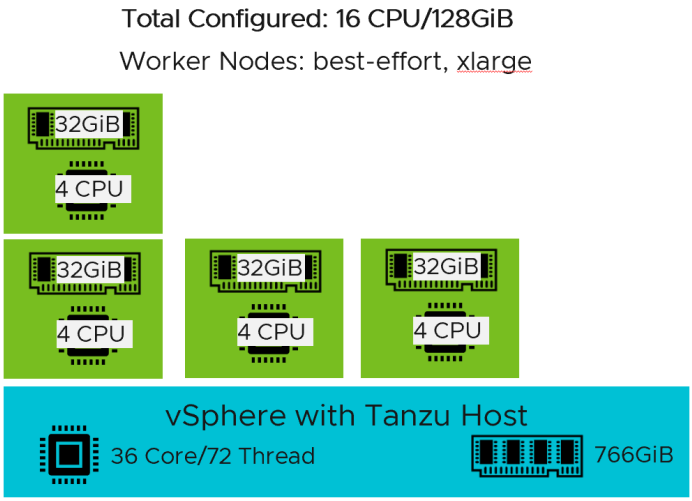

Figure 4. Right-sized cluster configured with knowledge of resource requirements

Using this cluster, the applications supported a total of 17,450 WvUsers, as shown in figure 5, below.

This process of examining resource requirements to avoid configuring excess resources for a cluster is often referred to as *right-sizing*. By right-sizing the cluster, we reduced overhead and improved performance by 8%. When the cluster is right-sized, we reduce idle workers to improve overall efficiency, and the workload also benefits from more co-located pods, resulting in better performance.

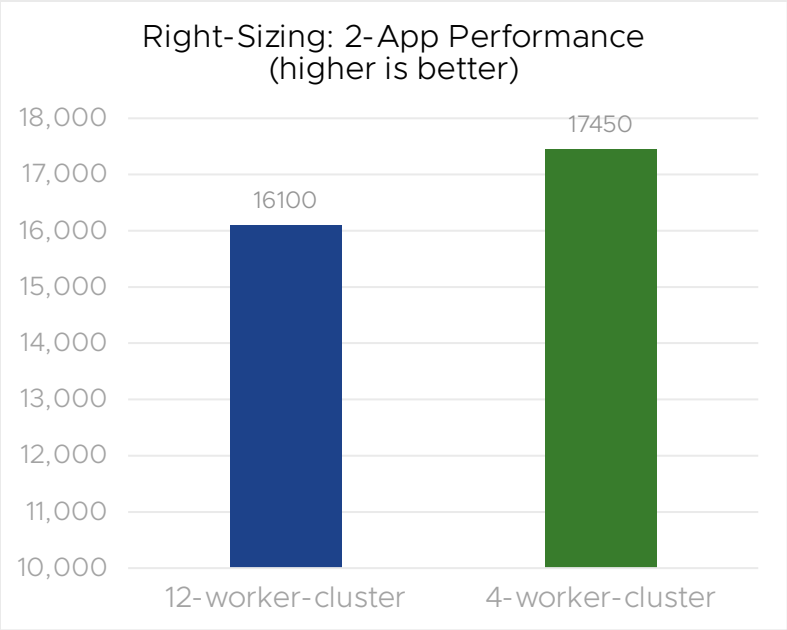

Figure 5. Performance benefit of right-sizing

Choose between guaranteed and best-effort VM classes

Problem description

When provisioning a Tanzu Kubernetes Grid cluster, you must select a virtual machine quality-of-service type for the control-plane and worker nodes. There are two options for the virtual machine quality-of-service type: guaranteed and best-effort. A node using a virtual machine class with type guaranteed has all its CPU and memory resources reserved by vSphere. For nodes with type best-effort, the CPU and memory resources are not reserved.

Note that the QoS types for TKG nodes are not related to the pod QoS classes discussed in the section Select a virtual machine size for worker nodes. Refer to that section for more information about pod QoS classes.

Best practices

Use guaranteed VM classes for critical workloads

When the TKG service provisions a node using a VM class of type guaranteed for a cluster, reservations are set for the CPUs and memory allocated for the VM.

- For CPUs, these reservations ensure that the ESXi scheduler prioritizes the node's VM when making decisions about which VMs to run. This gives the guaranteed nodes a performance advantage over the best-effort nodes. However, CPU reservations do not guarantee that the workload performance will not degrade when a vSphere cluster's hosts are experiencing high CPU utilization or the cluster's CPUs are over-committed. Sharing of host resources, such as CPU cache, memory bandwidth, and thermal frequency throttling can still degrade the performance of guaranteed nodes as over-commitment and utilization increase.
- For memory, these reservations ensure that the node's memory will not be swapped or ballooned when the ESXi host is experiencing high memory demand. This can have a significant performance benefit when the vSphere cluster's memory is over-committed due to other TKG clusters using best-effort VM classes or VMs with no memory reservations.

It is important to understand that nodes using the guaranteed VM class can only be powered on if there is an ESXi host in your vSphere cluster that has enough unreserved resources to satisfy the node's reservations. You cannot allocate more CPUs or memory for your guaranteed nodes than is available on your cluster. It is possible to allocate TKG clusters which use best-effort VM classes even on vSphere clusters whose resources are fully allocated to guaranteed nodes. The best-effort clusters will be able to use resources that are not used by the guaranteed clusters. However, over-committing the vSphere cluster CPUs in this manner can degrade the performance of the guaranteed clusters and should be done only with careful performance monitoring.

While using guaranteed nodes provides some hedge against resource deprivation by ensuring that these VMs get enough CPU time, they can still experience some performance degradation in cases of significant over-commitment. Please refer to the section Best Practices for Multiple Tanzu Kubernetes Grid Clusters for suggestions on how to choose the right level of overcommitment.

Use best-effort VM classes for resource over-commitment

When using nodes with a VM class of type best-effort, it is possible to over-commit the CPU and memory resources of a vSphere cluster. CPU resources are over-committed when the CPU resources allocated to VMs on the cluster exceed the CPU capacity of the hosts. Likewise, memory resources are over-committed when the memory allocated to VMs exceeds the memory capacity of the hosts.

Resource over-commitment is useful when you want to run multiple TKG clusters on a vSphere cluster and do not expect the individual clusters to fully use the resources allocated to their nodes at all times. The clusters will share the resources according to the normal vSphere resource management procedures. For more information, see the section CPU over-commitment using guaranteed and best-effort nodes.

Impact of load balancer resources

Problem description

When provisioning a cluster using the TKG service in VMware Cloud Foundation with VMware Tanzu with NSX-T networking, a small load balancer is provisioned in the NSX-T edge nodes. The resources available to this load balancer are limited, which in turn limits both the peak throughput through the load balancer and the number of Kubernetes LoadBalancer services it can support. When deploying a network-intensive workload in a TKG cluster or in many workloads which in aggregate have high network loads, this load balancer may become a bottleneck that impacts workload performance. Note that this document is correct as of vSphere 7.0. The association of TKG clusters to NSX-T load balancers may change in future releases. However, the best practice of monitoring for load balancer alarms in the NSX-T Unified Appliance will still apply.

Best practices

Check for load balancer alarms in NSX-T unified appliance

Kubernetes workloads that provision LoadBalancer services in a TKG cluster, either directly or as part of an ingress controller, may have their throughput and overall performance limited by the capabilities of the load balancer. For clusters provisioned with NSX-T providing load-balancing services, it is possible to determine if there is an issue by looking for alarms related to the load balancer in the NSX Unified Appliance console.

If the load balancer is a bottleneck, you must split your workloads among multiple TKG clusters to increase the available load balancer resources.

Demonstration: best practices in action

The importance of understanding the possible impact of the load balancer resources is shown by the results in figure 6, below. In this test, we initially ran six instances of the Weathervane benchmark application in a single TKG cluster with twelve best-effort xlarge worker nodes. On this cluster, a single instance of the application was able to support about 7,500 users, while six instances supported only 31,800. This was significantly lower than the 45,000 users that might have been expected when scaling up from one to six instances. An examination of the alarms within the NSX Unified Appliance for the cluster revealed a high number of alarms due to the exhaustion of load balancer resources. As a result, we reconfigured the system to run each application in a separate TKG cluster with two best-effort large worker nodes. Because each TKG cluster gets its own small load balancer allocated in NSX-T, this eliminated the bottleneck at the load balancer. We were then able to support 45,150 users on the six application instances, an improvement of over 40%.

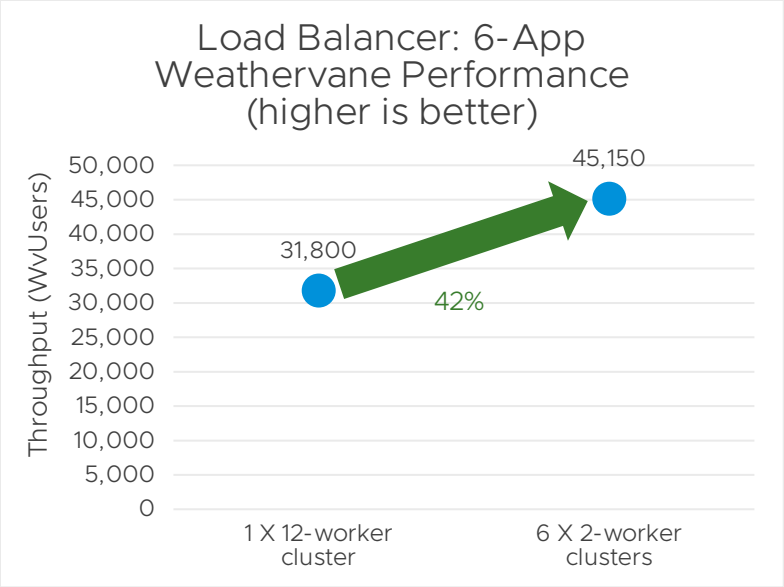

Figure 6. Performance benefit of reconfiguring to avoid load balancer bottleneck

Best Practices for Multiple Tanzu Kubernetes Grid Clusters

This section discusses performance-related considerations and best practices when deploying multiple TKG clusters on a single vSphere with Tanzu or VMware Cloud Foundation with Tanzu cluster. This information should be considered in addition to the best practices for individual TKG clusters whenever you are sharing the resources of a vSphere cluster among multiple TKG clusters.

vSphere resources shared among TKG clusters

Understand vSphere cluster CPU capacity

The CPU capacity of a vSphere cluster can be defined in two ways:

- The total number of physical cores on all servers in the cluster.
  - This definition is used in discussions of CPU over-commitment on a vSphere cluster. The CPUs of a vSphere cluster are said to be over-committed when the number of CPUs assigned to powered-on VMs is greater than the number of physical cores. CPU over-commitment is only possible if some VMs do not specify CPU reservations.
  - Example: Consider a vSphere cluster with four hosts, each with two CPU sockets. If the processors in each socket have 16 physical cores, then the cluster has a total capacity of 128 cores. If the total number of CPUs allocated to powered-on VMs is greater than 128, then the cluster is over-committed on CPUs.

- The total MHz for all physical cores on all servers in the cluster.
  - This definition is used by vSphere DRS when considering the impact of CPU reservations on admission-control decisions. The CPU reservation for a VM is specified in MHz. DRS will not power on a VM if its CPU reservation would cause the total reservations of all powered-on VMs to exceed the CPU capacity of the cluster. See the *vSphere Resource Management* documentation at https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-98BD5A8A-260A-494F-BAAE-74781F5C4B87.html for more information about vSphere DRS.
  - This definition is also used by vSphere high availability (HA) when considering the impact of CPU reservations on admission control decisions. Admission controls in vSphere HA ensure that sufficient failover capacity exists in a cluster. The default setting is to reserve the capacity equivalent, in MHz, of a single host. Note that this reduces the available CPU capacity on a cluster to even less than would be allowed by vSphere DRS's admission control policy.
  - Example: Consider a vSphere cluster with four hosts, each with two CPU sockets and 16 physical cores per socket. If the processors are running at 2.3 GHz, then the total CPU capacity of the cluster will be 4 * 2 * 16 * 2,300 = 294,400 MHz. If vSphere HA is enabled, as it must be to enable vSphere with Tanzu, then the capacity of one host will be reserved for failover, leaving 220,800 MHz available to satisfy CPU reservations from VMs.

Note that these definitions do not change when hyperthreading is enabled on the servers in the cluster. The performance benefit of hyperthreading is highly workload-dependent. As a result, the over-commitment definition and admission control decisions are always based on the number of physical cores and not the number of logical processors.

Understand vSphere cluster memory capacity

The memory capacity of a vSphere cluster is the sum of the memory configured on all hosts in the cluster and is specified in MBytes or GBytes. The memory capacity of the cluster is relevant to the following considerations:

- If the powered-on VMs on the cluster are allocated more memory than the capacity of the cluster, the cluster's memory is said to be over-committed. Over-committing memory can have a variety of impacts on the performance of the cluster and individual VMs. You should refer to the VMware technical whitepaper *Performance Best Practices for VMware vSphere 7.0* at https://www.vmware.com/techpapers/2020/vsphere-esxi-vcenter-server-70-performance-best-practices.html before over-committing memory on a vSphere cluster.
- vSphere DRS will not power on a VM if its memory reservation would cause the total reservations of all powered-on VMs to exceed the memory capacity of the cluster. See the *vSphere Resource Management* documentation at https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-98BD5A8A-260A-494F-BAAE-74781F5C4B87.html for more information about vSphere DRS.

- By default, vSphere HA reserves an amount of memory equivalent to the memory capacity of a single host to ensure sufficient failover capacity. vSphere HA will not allow a VM to be powered on if its memory reservation would cause the total reservations of all powered-on VMs to exceed this reduced memory capacity.

Over-committing vSphere cluster CPU with TKG clusters

CPU over-commitment basics

In the context of vSphere with Tanzu, a vSphere cluster is said to be over-committed on CPUs when the total number of CPUs assigned to nodes of TKG clusters, vSphere pods, and other VMs, is greater than the number of physical cores on all servers in the cluster. VMware vSphere uses sophisticated scheduling and resource management mechanisms to share the available CPU resources efficiently even when they are over-committed. See the *vSphere Resource Management* documentation at https://docs.vmware.com/en/VMware-vSphere/index.html and the *Performance Best Practices for VMware vSphere 7.0* guide at https://www.vmware.com/techpapers/2020/vsphere-esxi-vcenter-server-70-performance-best-practices.html for detailed information about these mechanisms.

The degree of CPU over-commitment is typically referred to using the ratio of the total number of allocated vCPUs to the number of available physical cores. For example, when running VMs with a total of 180 vCPUs on a vSphere cluster with 120 physical cores, the over-commitment ratio is 180/120, or 1.5 vCPUs per CPU. This ratio may be converted to a percentage when discussing the level of over-commitment. In the previous example, we would say that the level of over-commitment is 150%.

Over-committing CPU resources with TKG clusters may have the following benefits:

- For workloads that get a performance benefit from hyperthreading, over-committing CPUs can allow workloads to take advantage of all logical processors.
  - Example: A three-host cluster with a total of 120 physical cores will have 240 logical CPUs available when hyperthreading is enabled. Configuring TKG cluster nodes with a total of more than 120 vCPUs will allow workloads to take advantage of the additional logical processors.
  - Note that the performance benefit of hyperthreading is highly workload dependent. Most workloads will see less than a 30% improvement from hyperthreading.
- When running TKG clusters that have low CPU utilization, over-committing CPU by deploying additional TKG clusters on the same vSphere cluster will allow more efficient use of the server resources.
- When running workloads with cyclical CPU demands in TKG clusters, CPU can be over-committed by deploying additional TKG clusters to take advantage of otherwise idle resources during low-demand periods.

  - Make sure sufficient resources are available for workloads whose peak-demand periods overlap.
  - When running critical workloads, take care to avoid contention when over-committing CPUs. Consider using guaranteed VM classes for nodes and avoiding CPU over-commitment for hosts housing TKG clusters running critical workloads.

To determine the appropriate level of CPU over-commitment for your vSphere with Tanzu cluster, it is helpful to understand the following aspects of your TKG clusters:

- The expected average CPU usage of the workloads deployed in the cluster
- The expected peak CPU usage of the workloads deployed in the cluster
- The pattern of peak and low usage periods for the workloads deployed in the cluster

This data can be used to develop an initial plan for CPU over-commitment levels that avoid periods in which the combined resource usage of the clusters exceeds the capacity of the physical cores. Often you can obtain the CPU usage from performance monitoring data and history of past usage. If this data is not available, start with lower levels of CPU over-commitment and monitor the cluster for issues before adding additional TKG clusters.

To understand the performance impact of CPU over-commitment, it is critical to monitor not only the workload-level performance metrics, but also the vSphere metrics that are key indicators of over-commitment–related performance issues. The primary vSphere metrics that should be monitored when over-committing CPUs are Readiness, Ready time, or %ready if using esxtop. These are all different views of the same measurement, which is the percentage of time that a VM was ready to run but could not be scheduled due to contention for the physical CPUs.

The Readiness and Ready metrics are available in the performance charts provided by VMware vCenter. Figure 7 below shows the Advanced Performance chart for a vSphere host that is part of a vSphere with Tanzu cluster. This chart shows the Usage and Ready metrics over the previous day. During this period, there were several runs of the Weathervane benchmark with varying numbers of TKG clusters. The runs with higher usage levels had more TKG clusters and thus a higher level of CPU over-commitment. The runs with the highest usage also had a significant spike in Ready time, indicating possible performance problems due to the level of CPU over-commitment.

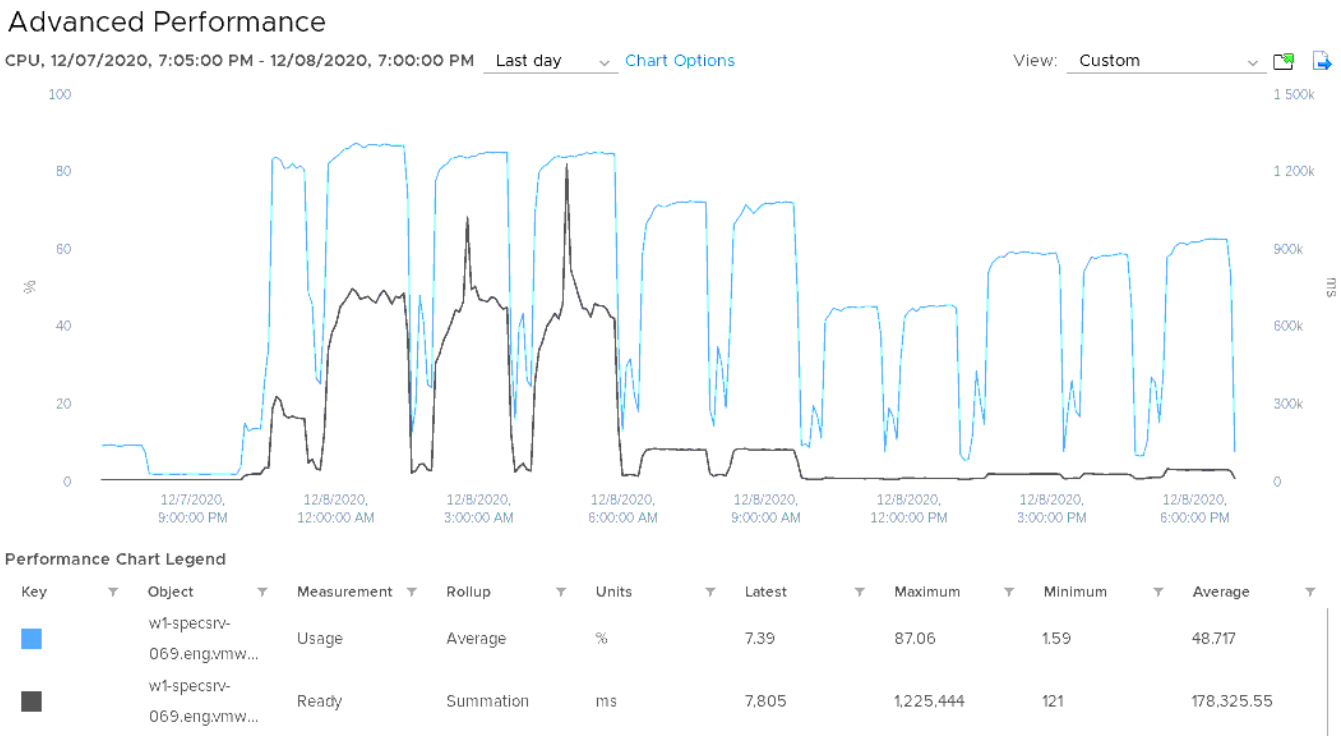

Figure 7. VMware vCenter CPU usage and Ready Time chart

CPU over-commitment using guaranteed and best-effort nodes

As discussed in the section Choose between guaranteed and best-effort VM classes, it is not possible to over-commit CPUs when using only guaranteed VM classes for your TKG nodes. The admission control policies of vSphere DRS and vSphere HA will limit the number of vCPUs allocated to guaranteed nodes to be less than the total number of physical CPU cores. To take advantage of the additional logical processors, as well as resources allocated to but not used by guaranteed nodes, it is necessary to deploy a combination of guaranteed and best-effort clusters. When over-committing with a combination of guaranteed and best-effort clusters, you should monitor for potential performance issues as discussed above.

Demonstration: best practices in action

Improve resource utilization with vSphere over-commitment

With the ability to over-commit CPU and memory resources with TKG clusters in vSphere, you can significantly improve the resource utilization of vSphere clusters. This is particularly true when the resource requests assigned to pods overestimate the actual resource usage, or when the pods in one Kubernetes cluster cannot take advantage of all the available node resources.

This effect is demonstrated by the measurements shown in figure 8. In this example, we used a single ESXi host with 40 physical cores and 80 logical CPUs. We gradually increased the level of CPU over-commitment by adding TKG clusters. Each TKG cluster had two best-effort medium

worker nodes and one best-effort small control-plane node, for a total of 6 CPUs. In each cluster, we ran one instance of the Weathervane xsmall application configuration, using the findMax run strategy to find the maximum load supported by the cluster.

Figure 8 below shows the maximum number of Weathervane users that could be supported as we increased the number of TKG clusters, and thus the level of CPU over-commitment. Because the Weathervane application does not use the full CPU capacity of the worker nodes and the control-plane nodes are largely idle, a significant benefit was obtained by over-committing the host's CPUs. In this case, the peak load was achieved with 20 TKG clusters, which represents a 300% over-commitment of the available CPU cores.

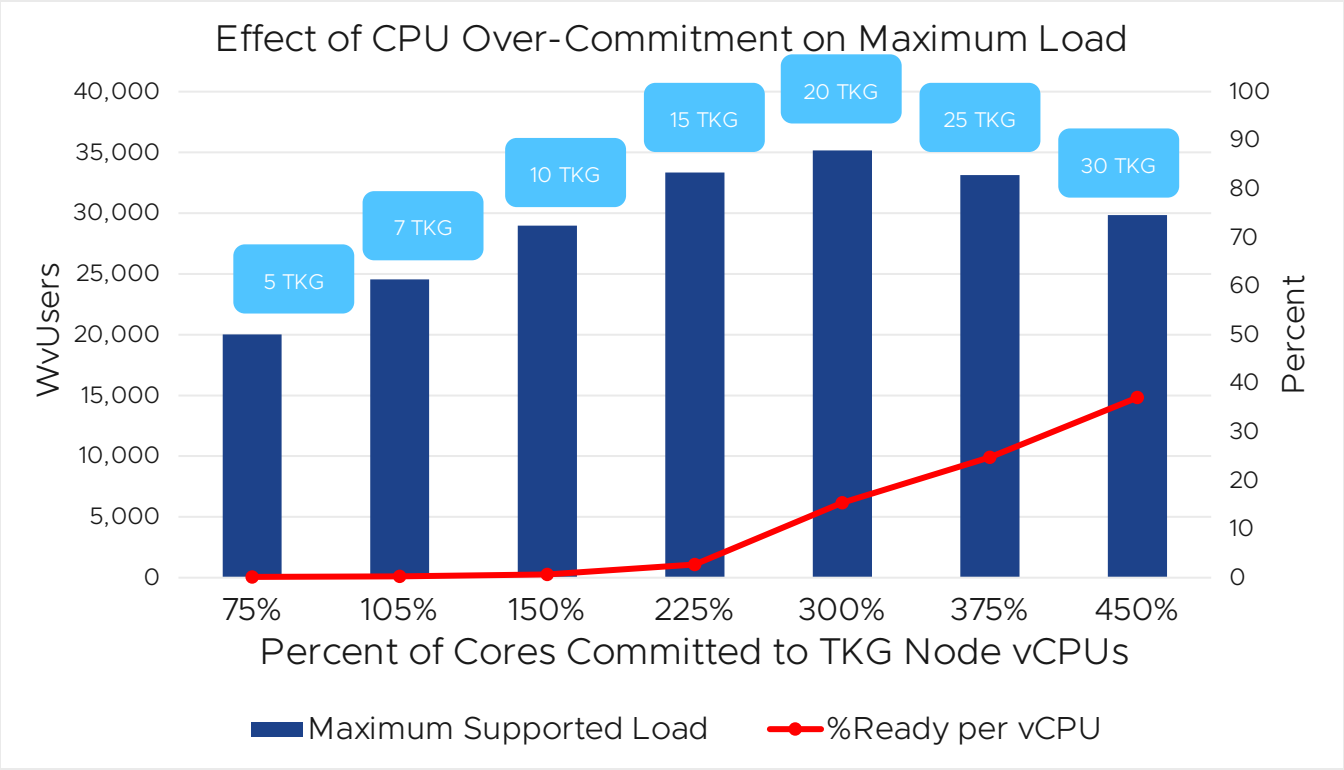

Figure 8. Performance impact of CPU over-commitment

Use metrics to monitor impact of CPU over-commitment

The chart in figure 8 also demonstrates how monitoring readiness can provide advanced warning of performance issues related to CPU over-commitment with TKG clusters. As stated above, the peak load was achieved with 20 TKG clusters. However, the improvement from 15 to 20 clusters was relatively small and came with a marked increase in readiness. As we added more TKG clusters, the peak load decreased due to contention and additional scheduling overhead. The readiness or %ready metrics capture this contention and indicate that the level of CPU over-commitment should be decreased to improve the performance of the deployed workloads.

Guaranteed vs best-effort

The chart below shows the results of running the Weathervane benchmark in TKG clusters with four xlarge worker nodes running on a host with 32 physical cores. Each TKG cluster was allocated a total of 16 CPUs and ran two Weathervane application instances which requested a total of 9.6 CPUs. In configurations with one and two TKG clusters, the guaranteed clusters slightly out-performed the best-effort clusters. It was not possible to allocate more than two guaranteed clusters on this host due to insufficient remaining unreserved CPU and memory resources. However, it was possible to add more best-effort clusters. The peak throughput for the Weathervane benchmark was achieved with four TKG clusters, which represented a 200% over-commitment of the host's CPU resources to the TKG clusters. The application instances running on the four TKG clusters requested a total of 38.4 CPUs.

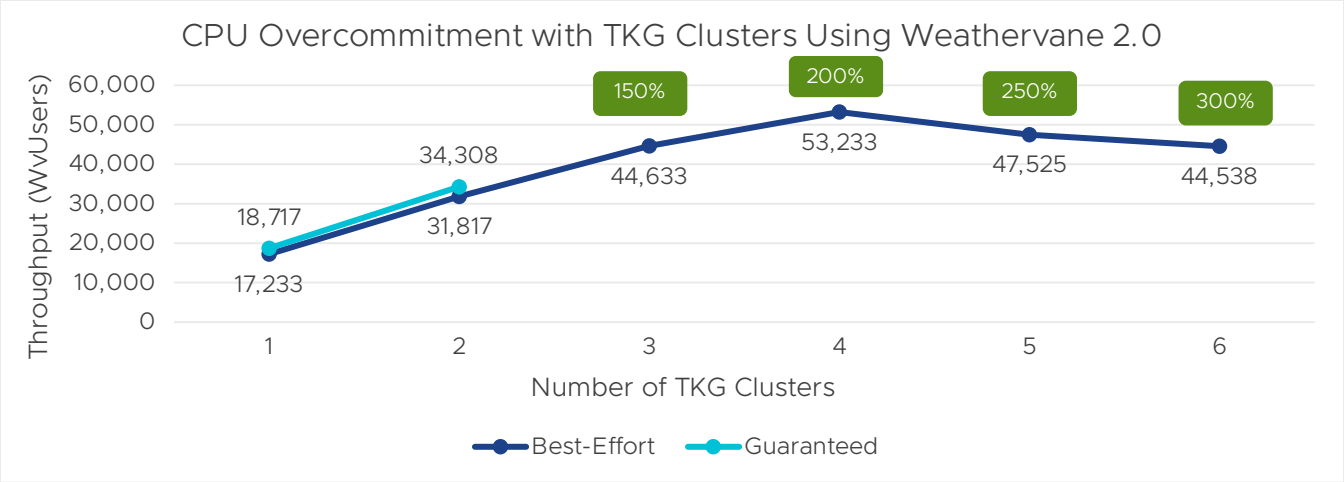

Figure 9. Performance benefit of CPU over-commitment with best-effort clusters

Best Practices for Deploying Workloads in vSphere Pods

Specifying container resource requests and limits

Problem description

When a workload is deployed using vSphere pods on the supervisor cluster, each pod is deployed using a Container Runtime for ESXi (CRX), which is essentially a lightweight VM. The CPU and memory configuration of a CRX VM is determined based on the resource requests and limits of the containers in the pod deployed in that CRX. The number of CPUs assigned to the CRX of a vSphere pod imposes a fundamental limit on the maximum CPU time that pod can receive. As a result, the performance impact of specifying resource requests and limits for vSphere pods differs from using those same limits on Kubernetes pods deployed on other platforms.

Best practices

Understand how resources are assigned to a CRX VM

The CPU resources assigned to the CRX VM for a vSphere pod are determined as follows:

- If no CPU requests or limits are specified for any containers in the pod, then the CRX VM is assigned 1 vCPU.
- If CPU requests are set for any containers in the pod, but no limits are specified, then the CRX VM is assigned a number of vCPUs equal to the sum of the CPU requests of all of the containers, rounded up to the nearest full CPU.
- If both CPU requests and limits are specified for any containers in the pod, then the CRX VM is assigned a number of vCPUs equal to the sum of all of the limits, rounded up to the nearest full CPU. A vSphere-level CPU limit is then placed on the CRX VM to prevent it from using more CPU than specified by the container-level limits.

The CRX VM will have a vSphere-level reservation for CPU resources equivalent to the sum of the container-level CPU requests. Because vSphere CPU reservations are set in MHz, the container-level requests will be converted to MHz. A CPU request for 1 CPU in a container specification is equivalent to a reservation equal to the clock frequency of a single physical core. The implications of this reservation on admission control for vSphere pods is discussed below.

The memory resources assigned to the CRX VM for a vSphere pod are determined as follows:

- If no memory requests or limits are specified for any containers in the pod, then the CRX VM is assigned 1 GiB.
- If memory requests are set for any containers in the pod, but no limits are specified, then the CRX VM is assigned memory equal to the sum of the requests.

- If both memory requests and limits are specified for any containers in the pod, then the CRX VM is assigned memory equal to the sum of all the limits, except that the memory assigned will be no larger than either 3x the sum of the memory requests or the largest memory limit of any container in the pod, whichever is larger.

The CRX VM will have a vSphere-level reservation for memory resources equivalent to the sum of the container-level memory requests. The implications of this reservation on admission control for vSphere pods is discussed below.

Set resource requests and limits for vSphere pod

The rules for assigning resources to CRX VMs have the following implications for the performance of vSphere pods:

- It is critical to specify resource requests or limits for all containers that will run in vSphere pods.
  - vSphere pods whose containers do not specify CPU and memory requests or limits will run in CRX VMs with only 1 vCPU and 1 GiB of memory. While this may be sufficient for some pods, it is likely to seriously constrain the performance of many workloads and may prevent containers from starting.
- Specifying resource requests without limits causes the requests to act as an upper-bound on the resources that can be used by a pod.
  - This is very different than the behavior of Kubernetes pods on other platforms, where a pod with no limits specified can use node resources beyond its requests as long as they are not being used by other pods.
  - As a result, if no limits are to be specified, the requests assigned to containers in a vSphere pod must account for peak resource needs. Note that because requests are translated into vSphere-level reservations, this has implications for admission control and the number of vSphere pods that can be deployed.
- Specifying both resource requests and limits allows the CRX VM to be sized for worst-case usage, while only reserving the resources needed for typical resource demands.

As a result, it is highly recommended that both resource requests and limits be specified appropriately for containers in vSphere pods. The requests should correspond to typical usage, while the limits should correspond to expected peak demands. Note that limits should not be set higher than necessary because provisioning VMs with more resources than needed can adversely affect performance.

Demonstration: best practices in action

The results in figure 10 below show the impact of resource request and limits on the performance of the Redis Memtier benchmark. The benchmark was run in a single vSphere pod with different resource requests and limits. The three cases shown in this chart are:

1. No CPU request or limit was set for the vSphere pod.

In this case, the CRX VM was assigned only 1 vCPU. This limited performance to 89,000 transactions per second (TPS) as shown in the left-most datapoint in figure 10 below. Note that this limitation is different than would be experienced by the same pod in a TKG cluster, where the pod would be able to access all available resources of its Kubernetes node.

2. The CPU request was set to 2, but the limit was not set.

   In this case the CRX VM was assigned 2 vCPUs, which almost doubled the performance to 175,000 TPS as shown in the datapoint at the center of figure 10. However, the pod was still limited by the available CPU, and it was not able to access the remaining node resources.

3. The CPU request was set to 2, and the limit was set to 8.

   In this case, the CRX VM was assigned 8 vCPUs, with the capacity of only two of those CPUs being reserved in vSphere. This allows the performance to increase to 252,000 TPS, a value that is no longer limited by the available CPU resources as shown in the right-most datapoint in figure 10.

While the performance would have been equivalent if the CPU request had been set to 8, this would have reserved CPU resources that would not have been used by the pod, thus preventing other vSphere pods from being deployed to use those resources.

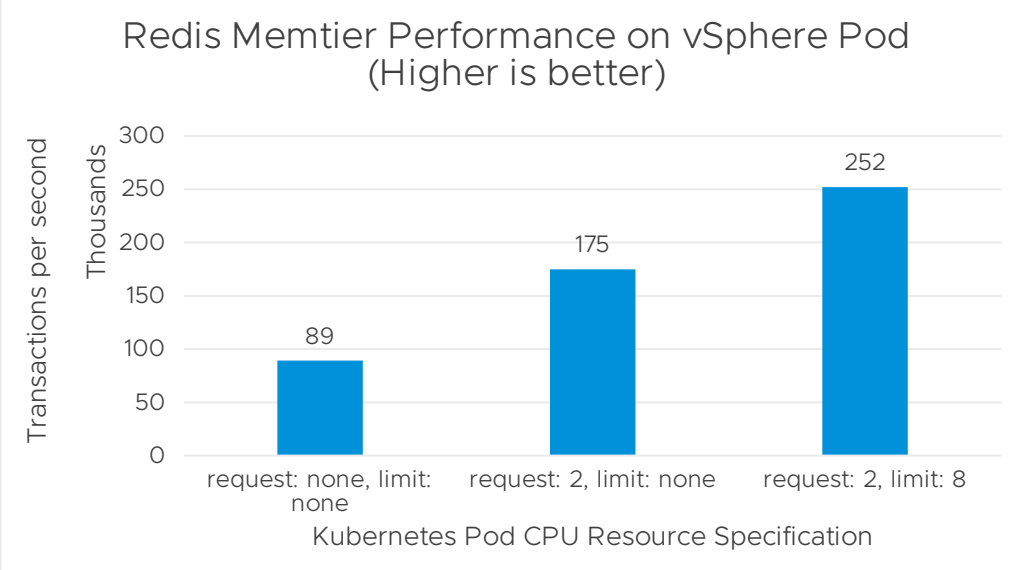

Figure 10. Impact of requests and limits on vSphere pod performance

Admission control for vSphere pods

Problem description

When deploying workloads as vSphere pods, the CPU and memory requests specified for the pods result in vSphere-level resource reservations on the CRX VMs. These reservations guarantee resources for the pods, but also limit the number of vSphere pods that can be deployed on a cluster. In order to understand how many vSphere pods can be deployed on a cluster, you must understand both vSphere DRS and vSphere HA admission control policies. These policies also affect combinations of vSphere pods and TKG cluster nodes using the guaranteed VM class.

Best practices

Understand vSphere reservations and admission control

When deploying vSphere pods with resource requests, the corresponding resources are reserved in vSphere for the CRX VM. CPU resource requests are translated into VM-level CPU reservations, while memory requests are translated into VM-level memory reservations. These reservations are used by vSphere DRS and vSphere HA to determine whether the VM can be powered-on on any host in the cluster, a process referred to as admission control. All vSphere with Tanzu clusters must have vSphere DRS and vSphere HA enabled, thus requiring all vSphere pods and TKG node VMs to pass the admission control tests. If the VM fails any of the admission control tests, it will not be powered on and the user will be notified of the failure event.

For vSphere, memory reservations are specified in GB or MB, and CPU reservations are specified in MHz. For admission control purposes, the total CPU capacity of an ESXi host is the base frequency of the processors in MHz multiplied by the number of physical cores. Note that hyper-threads or the extra CPU capacity from turbo boost are not counted in the CPU capacity. A Kubernetes-level CPU reservation of one CPU is translated into the MHz equivalent of one physical core. For example, a host with 16 cores running at 2,600MHz will have a total capacity of 41,600MHz to allocate for VM-level CPU reservations. If a vSphere pod has a total of Kubernetes CPU requests equal to 750 mcores, or 75% of a CPU, then the VM-level CPU reservation will be 0.75 * 2,600MHz or 1,950MHz. The additional CPU capacity represented by hyperthreads or turbo boost cannot be reserved but can be used by best-effort TKG worker VMs or by vSphere pods with CPU limits that are higher than their requests.

vSphere DRS will only allow a VM to be powered on if at least one host in the cluster has available resources to satisfy the CPU and memory reservations of the VM. The available resources are those that are not already allocated to reservations associated with a powered-on VM.

vSphere HA reserves resources as failover capacity to allow VMs to be powered on elsewhere in the cluster in the event of a host failure. By default, vSphere HA reserves the capacity equivalent of a single cluster host. vSphere HA will not allow a VM to be powered on if its reservations would cause available resources in the cluster to fall below the reserved failover capacity, even if a host in the cluster has sufficient resources under the vSphere DRS admission control policies. As a result,

the total reservations of vSphere pods will be constrained to be one host less than the full capacity of the cluster.

Impact of load balancer resources

Problem description

When the VMware Tanzu Basic (with vSphere) Workload Management Platform is configured with NSX-T Data Center as the networking stack, a medium load balancer is provisioned in the NSX-T edge nodes. This load balancer supports Kubernetes LoadBalancer services provisioned for vSphere pods. The resources available to this load balancer are limited, which in turn limits both the peak throughput through the load balancer and the number of Kubernetes LoadBalancer services it can support. While the medium NSX-T load balancer supports more traffic and LoadBalancer services than the small load balancer allocated for a TKG cluster, it can still become a bottleneck in configurations with high network load or large numbers of LoadBalancer services. Note that this document is correct as of vSphere 7.0. The association of supervisor clusters and namespaces to NSX-T load balancers may change in future releases. However, the best practice of monitoring for load balancer alarms in the NSX-T Unified Appliance will still apply.

Best practices

Check for load balancer alarms in NSX-T unified appliance

Kubernetes workloads that provision LoadBalancer services in the Supervisor cluster, either directly or as part of an ingress controller, may have their throughput and overall performance limited by the capabilities of the NSX-T load balancer. It is possible to determine if there is an issue by looking for alarms on the load balancer in the NSX Unified Appliance console.

If the load balancer is a bottleneck, split your workloads among multiple VMware Tanzu Basic clusters to increase the available load balancer resources.

Conclusion

The information and best practices discussed in this document will help you to obtain good performance when deploying workloads on vSphere with Tanzu and VMware Cloud Foundation with Tanzu. The key messages are:

- Understand the resource needs of your workloads and reflect them as accurately as possible in the resource requests of your pods.
- Monitor the actual resource usage of your workloads, and adjust the resource requests, node sizes, and node count to maintain good performance over time.
- Use guaranteed VM classes when deploying critical workloads in TKG clusters.
- When appropriate, use best-effort VM classes to over-commit CPU with TKG clusters to efficiently utilize the cluster resources.
- When deploying network-intensive workloads on VMware Tanzu Basic with NSX-T networking, monitor for alarms on the NSX-T load balancer and, if necessary, use multiple TKG clusters to distribute load over multiple load balancers.
- When deploying workloads in vSphere pods, it is critical to specify resource requests and limits to ensure that the CRX VM backing the pod is allocated sufficient resources. The interpretation of requests and limits has key differences from their use in other Kubernetes clusters, and must be understood to achieve the best possible performance.

About the Authors

Hal Rosenberg is a staff-2 performance engineer at VMware. His focus areas are Kubernetes performance, benchmark development, and performance troubleshooting. He is the architect and lead developer for the Weathervane open-source performance benchmark.

Karthik Ganesan is a staff-2 performance engineer at VMware R&D with deep expertise in cloud systems performance. He received his PhD in Computer Engineering from the University of Texas at Austin, holds multiple patents, authored numerous research papers and articles, and gives technical talks at reputed conferences.

Benjamin Hoflich is a performance engineer at VMware. He is experienced with investigating and improving the performance of software products for scalability, response times, and throughput, along with the development of benchmarks and workloads.