Enterprise Workload Performance on Kubernetes with CPU Limits

VMware vSphere 7 with Tanzu

Performance Study - October 5, 2021

vmware®

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

Table of Contents

Introduction

Executive Summary

Kubernetes is rapidly becoming the deployment platform of choice for enterprise applications. However, many misconceptions exist about the performance implications of running these applications on virtualized infrastructures compared to running directly on bare-metal hosts. This paper uses the open-source Weathervane 2 benchmark to compare the performance of an enterprise web application running on Kubernetes clusters deployed on VMware vSphere® 7.0 Update 2 and on a bare-metal server.

The results show that the performance of Kubernetes clusters deployed on vSphere with the VMware Tanzu® Kubernetes Grid™ (TKG) service can significantly exceed that of bare-metal Kubernetes clusters, particularly when using Kubernetes CPU limits on modern, multi-core hosts. The Weathervane benchmark uses Kubernetes CPU limits as part of its core design, and those limits have a dominant impact on the configurations and results in this paper.

Initial performance results on the bare-metal cluster were poor as a result of the over-throttling of pods with the implementation of Kubernetes CPU resource limits and its behavior on large bare-metal hosts. Those initial results are included in the lower sections of figure 5. In order to improve the bare-metal results, several adjustments were made, including the enablement of Kubernetes CPU Manager policies and the modification of the workload to create a new Weathervane benchmark configuration size. Details are explained below in the Tuned Bare-Metal Results section.

Even after the significant adjustments to improve the bare-metal results, the resulting comparison shows the vSphere cluster significantly outperformed the bare-metal cluster.

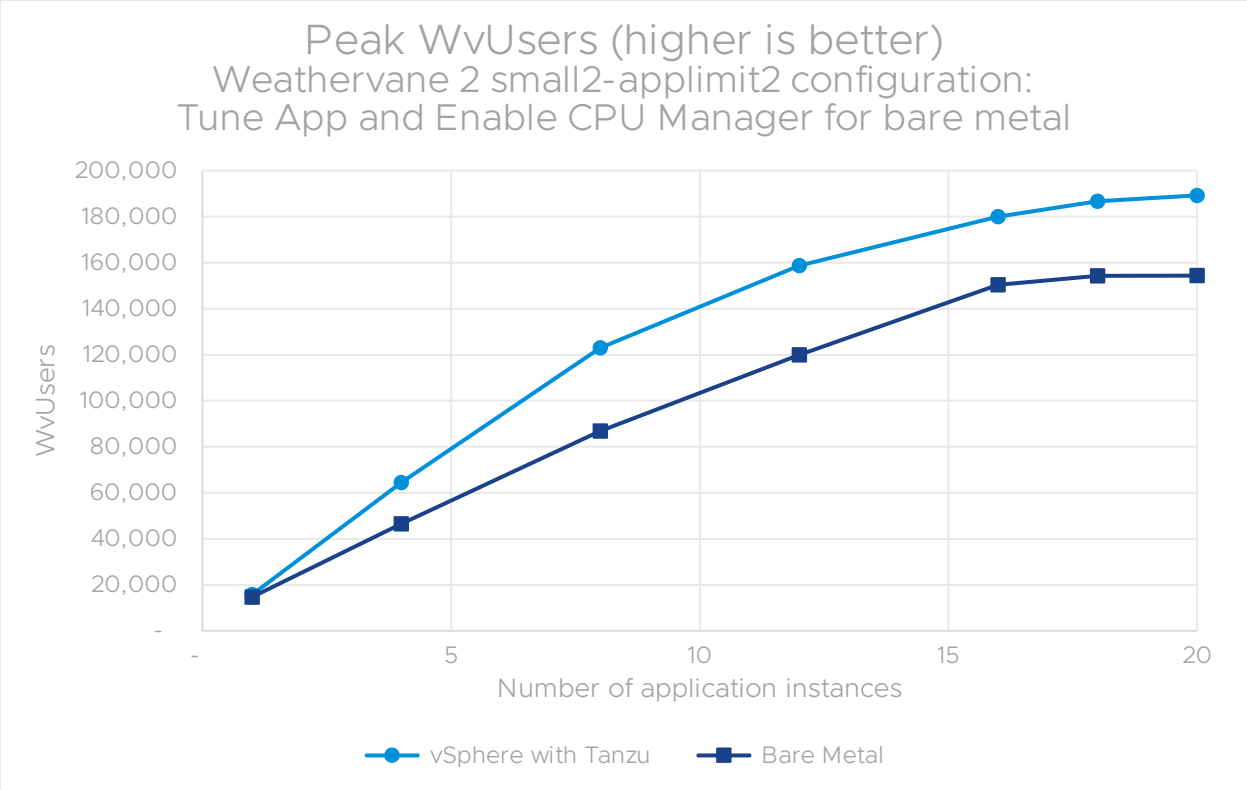

Figure 1. Results with bare-metal CPU Manager enabled and small2-applimit2 configuration size

A further investigation of these results showed that poor CPU assignment decisions by the Kubernetes CPU Manager on bare metal were at least partly responsible for the performance gap. To get the best possible scores from the bare-metal cluster, additional tests were run with simultaneous multi-threading (SMT) turned off in the BIOS for the bare-metal server. The results from these tests are shown in the green/diamond line in figure 2 where the bare-metal cluster performed equivalently to the vSphere cluster at up to sixteen application instances. However, disabling SMT reduced the number of CPUs available to satisfy the CPU resource requests of the Weathervane application instances, thus limiting the peak performance to about 5% lower than the vSphere cluster.

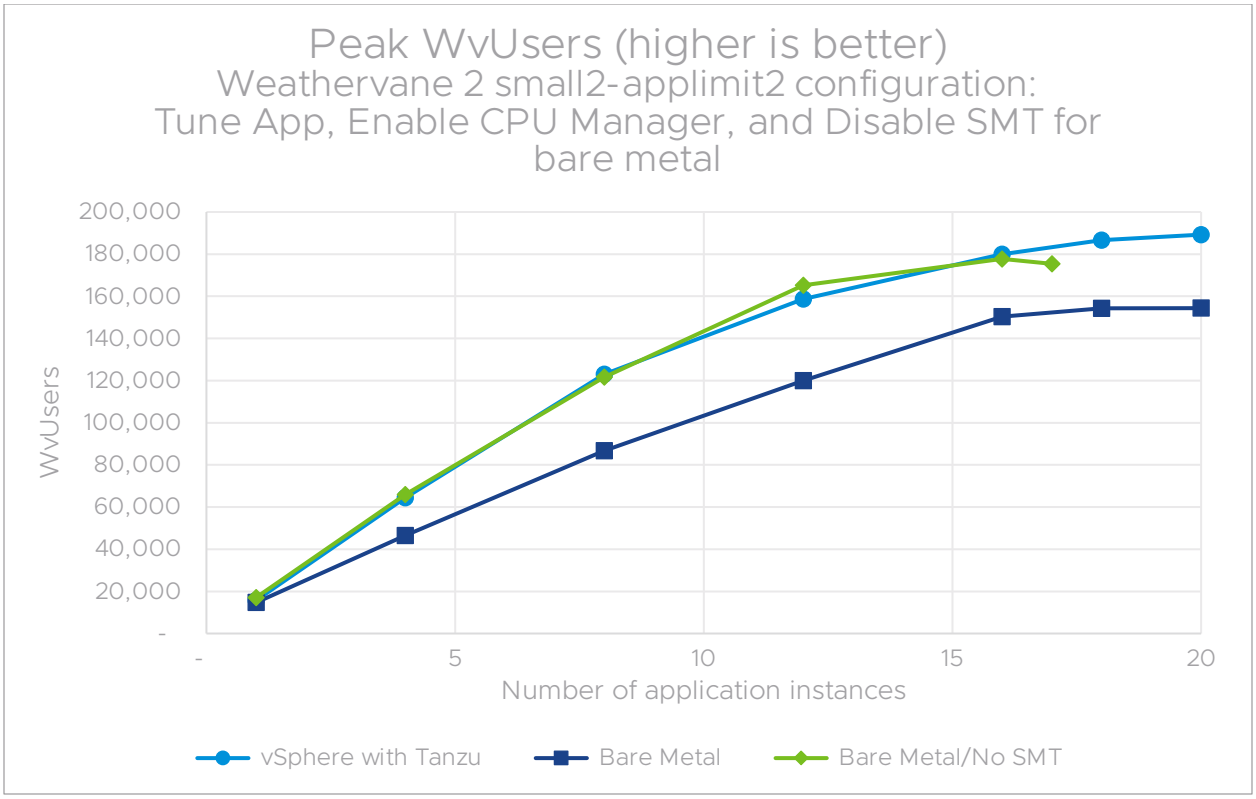

Figure 2. Results with SMT disabled on the bare-metal cluster

On vSphere with Tanzu, it is possible to achieve excellent performance without extraordinary tuning and configuration efforts when running enterprise applications with Kubernetes CPU limits on Kubernetes clusters. By using vSphere, you take advantage of virtualization benefits, including an elastic platform that lets you easily scale up with vCPUs or VMs, vMotion, DRS, and HA, to name a few.

Intended Audience

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

Related Documents

For performance best practices specific to Kubernetes on vSphere and vSphere with Tanzu, refer to the VMware technical whitepaper Performance Best Practices for Kubernetes with VMware vSphere with Tanzu and VMware Cloud Foundation with Tanzu.

For general performance best practices for vSphere 7, refer to the VMware technical whitepaper Performance Best Practices for VMware vSphere 7.0.

Test Environment

Test Workload

The data in this paper was generated using the Weathervane 2 performance benchmark. Weathervane 2 is an open-source benchmark that lets admins and engineers compare 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. The application can be deployed in multiple pre-tuned configurations which vary the number of replicas of each service, the resource requests and limits for the services, and the amount of data used in each run. The tests documented in this paper use the *small2* and the later added *small2-applimit2* configuration sizes. To scale up load on the clusters under test, multiple instances of the application can be configured to run simultaneously with the unified workload driver. This can be thought of as modeling a multi-tenant environment, with each independent application instance constrained by a specified amount of Kubernetes resource requests and limits.

The workload driver generates load on the clusters by simulating users interacting with the application instances. The performance metric for 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.

Weathervane uses Kubernetes CPU limits as part of its core design. Specifying CPU limits for the Tomcat pod ensures that it becomes the bottleneck for each application instance. The resulting workload typically bottlenecks on a characteristic of the system under test, such as CPU. Without

this limit, the workload would bottleneck on an internal limitation of the Weathervane application, such as the individual tuning of each service component with unbounded CPU availability.

Weathervane supports several configuration parameters to allow customizing tests for specific testing goals. We used the defaults for all non-required parameters. Weathervane does require the user to specify the type of Kubernetes service to be used for ingress to the application from the workload drivers. The options are LoadBalancer, NodePort, and ClusterIP Services. LoadBalancer services can only be used with clusters which support provisioning external IP address for these services. For the tests documented in this paper, NodePort services were used.

Additional information about Weathervane 2 can be found at https://github.com/vmware/weathervane.

Testbed Configuration

The tests discussed in this paper were run on the testbed depicted in figure 3. Four identical Dell PowerEdge R7525 physical servers were used with the following configuration:

- Two AMD EPYC 7742 CPUs running at 2.25GHz. The CPUs each had 64 cores, each with two SMT threads, for a total of 256 logical processors
- 2TB of memory
- Storage: Three 1.75TB SSDs and one 1.46TB NVMe devices were used on each server.
  - The servers running vSphere used this storage to form a vSAN datastore.
  - The bare-metal server used this storage to provide persistent volumes to pods using Rook/Ceph, as discussed below.
- Network: 40Gbps Ethernet

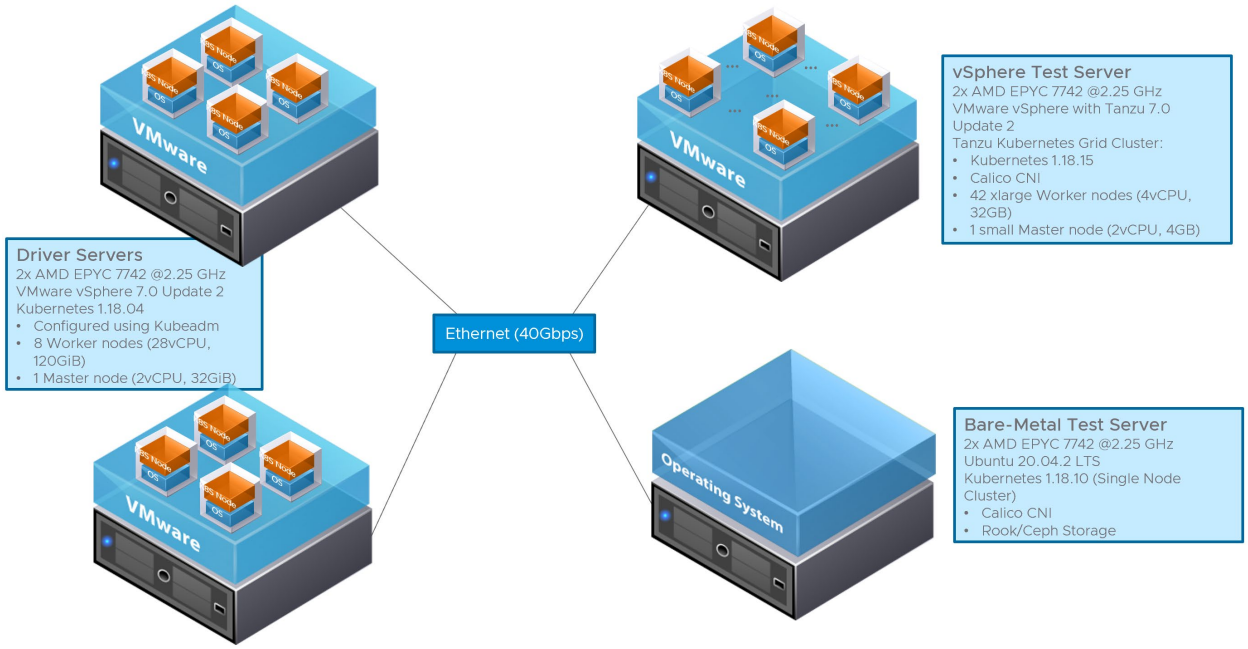

Figure 3. Testbed diagram

The two Test Servers hosted independent Kubernetes clusters.

- One server hosted a Tanzu Kubernetes Grid (TKG) cluster running on vSphere 7.0 U2.
- One server hosted a single-node, bare-metal Kubernetes cluster.

The two Driver Servers hosted another Kubernetes cluster used to drive load to the Test Servers.

- Two servers hosted a Kubernetes cluster running on vSphere 7.0 U2 for the workload driver pods.

The details of the clusters on the Test Servers are discussed below.

TKG Cluster

On the vSphere test server, a Kubernetes cluster was created using the Tanzu Kubernetes Grid (TKG or TKGs) service. This cluster will be referred to as the TKG cluster in the remainder of this document.

The TKG cluster had the following configuration:

- vSphere with Tanzu, Version 7.0 Update 2
- Kubernetes Version 1.18.15
- 42 worker nodes, each with 4 vCPUs and 32GiB of memory (TKG best-effort-xsmall VM class)
- 1 control plane node with 2 vCPUs and 4GiB of memory (TKG best-effort-small VM class)

- Networking: Calico was used for the Kubernetes networking overlay, because it had been used historically in this setup
- Storage: Persistent volumes were allocated from a vSAN datastore using one NVMe as a caching device and three SSDs as capacity devices

Bare-Metal Cluster

On the bare-metal test server, a Kubernetes cluster was created using the kubeadm tool. This cluster will be referred to as the bare-metal cluster in the remainder of this document.

The bare-metal cluster had the following configuration:

- OS: Ubuntu 20.04.2 LTS
- Kubernetes version 1.18.10 (vanilla edition)
- This cluster was deployed as a single-node cluster, with the node acting as both the control plane node and worker node. To run workloads on the node, the master taint was removed with the command:

```
kubectl taint nodes --all node-role.kubernetes.io/master-.
```

- The following customizations were made to the kubelet on the bare-metal node:

```
cpuManagerPolicy: static
systemReserved:
  cpu: 200m
  memory: 1Gi
kubeReserved:
  cpu: 1000m
  memory: 32Gi
maxPods: 500
```

  - The settings for *cpuManagerPolicy*, *systemReserved*, and *kubeReserved* were used to enable the Kubernetes CPU Manager.
  - The change to *maxPods*, from the default of 110, was made to allow sufficient pods to be run to max out the resources on the node.

- Networking: Calico was used for the Kubernetes networking overlay
- Storage: Persistent storage for the bare-metal cluster was provided using Rook/Ceph. Data was stored on four OSDs, each on its own SSD device, with a separate NVMe device used as a block db device.
- The following tunings were made at the OS level to improve the performance of the cluster or its ability to support the required resources:
  - Increased the number of transmit/receive descriptor rings for the Ethernet adapter in order to get better distribution of network interrupts

```
ethtool -L eno33np0 combined 74
```

- Increased the maximum number of processes (nproc) to 8388608
- Increased the maximum number of open files (nofile) to 8388608

Baseline Results

The first tests performed for this study used Weathervane with the small2 application configuration. The goal was to compare the performance of the TKG cluster with the bare-metal cluster at a variety of loads. The load was varied by scaling up the number of small2 instances and comparing the WvUsers achieved by each cluster for each configuration. The number of instances was increased until there was no further improvement in total performance, or until no additional instances could be accommodated by the CPU and memory resources of the clusters.

Figure 4 shows the application pods and their CPU/memory requests for the small2 configuration. Each application instance consists of eight pods, with total CPU requests of 4.8 CPUs and total memory requests of 31.84 GiBytes. The Tomcat pod defines both resource requests and limits of 1.5 CPUs and 5 GiBytes. Specifying CPU limits for the Tomcat pod ensures that it becomes the bottleneck for each application instance.

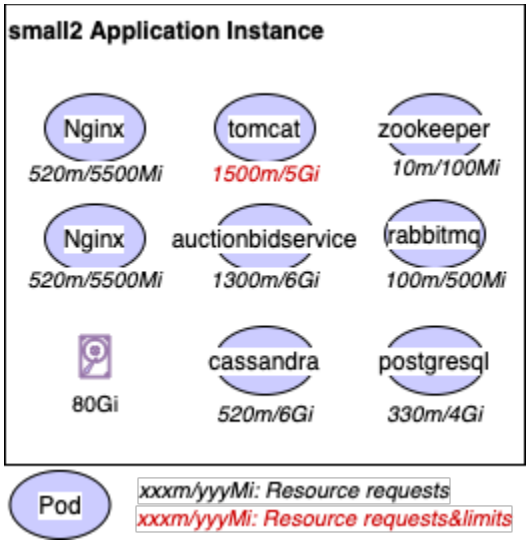

Figure 4. Weathervane small2 configuration

In these tests, the number of application instances of the small2 configuration was increased from 1 to 28, at which point the CPU utilization approached saturation.

Results

Figure 5 shows a summary of the results obtained by running the Weathervane small2 configuration on the vSphere and the bare-metal clusters. In these tests, the TKG cluster significantly outperformed the bare-metal cluster. The TKG cluster maxed out at almost 190,000 WvUsers at 28 application instances, while the bare-metal cluster could only support about 36,000

WvUsers at 18 application instances. After 18 instances, the bare-metal cluster was unable to run successfully.

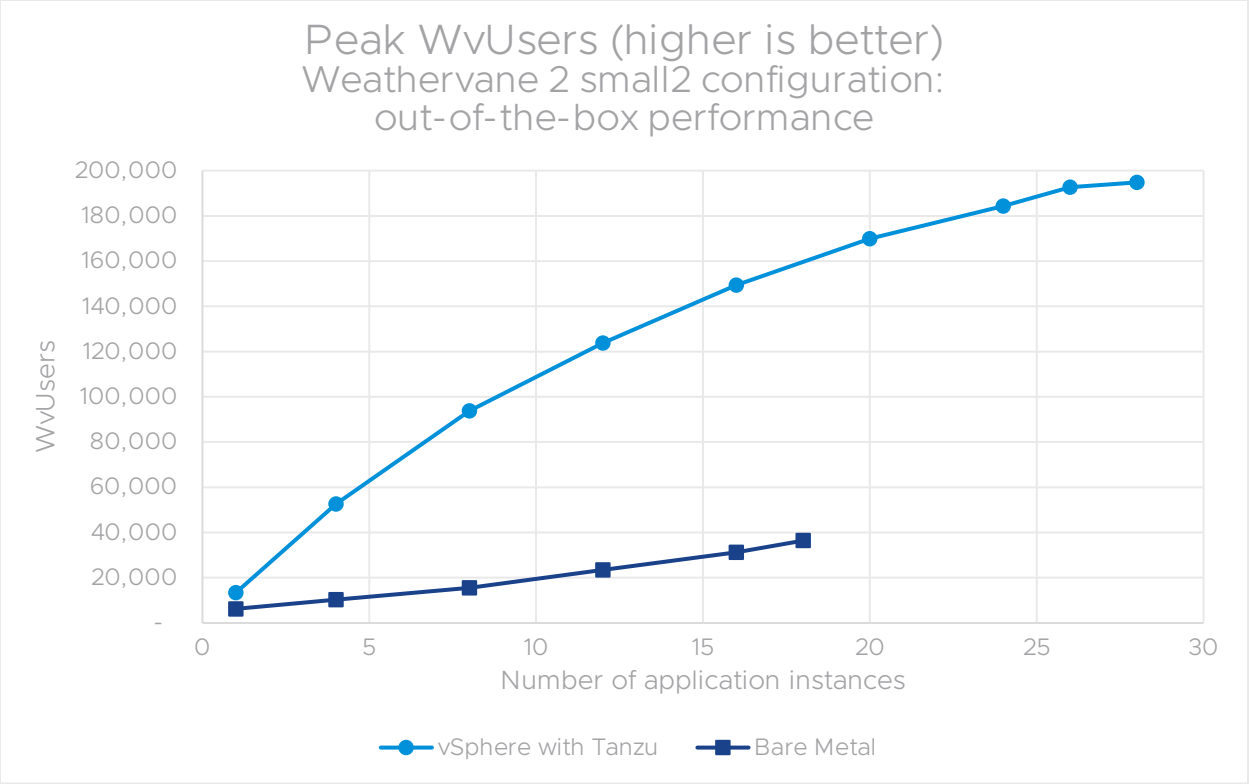

Figure 5. Baseline performance results

Discussion

The poor performance on the bare-metal cluster was unexpected. A root-cause investigation determined that the results were due to over-throttling of the application server pods, most likely due to the implementation of CPU limits in the Linux kernel combined with characteristics of both the application and the underlying hardware. Details of this over-throttling are discussed in this section. Efforts made to work around the problem and improve the performance of the bare-metal cluster are discussed in subsequent sections.

CPU limits specified for Kubernetes pods are enforced in the Linux kernel using the Completely Fair Scheduler (CFS) bandwidth control implementation (https://www.kernel.org/doc/html/latest/scheduler/sched-bwc.html). The amount of CPU used by containers in a pod is limited by dividing time into periods and assigning a run-time quota for each period. The quota is proportional to the CPU limit specified for the container. The default period is 100ms. In the case of the application server pod in the Weathervane small2 configuration, the limit for the Tomcat container is 1.5 CPUs, or 1500m cores. As a result, the container is given a quota of 150ms of run-time in each 100ms period. The quota can be larger than the period because threads from the container can be running on multiple CPUs. As soon as all threads of the Tomcat process

have run for 150ms in a period, summed over all CPUs in the server, the container is throttled until the start of the next period.

If a container is not able to use its entire compute quota due to the operation of CFS bandwidth control, then it is said to have been over-throttled. Figure 6 shows the average time that the Tomcat container was throttled by the CFS bandwidth control algorithm for both the TKG and bare-metal Kubernetes clusters. This data was collected during runs with eight small2 application instances, corresponding to the eight application instance points in figure 5. The bare-metal run had a shorter duration due to stopping at a smaller WvUsers result. This chart shows that the container on the bare-metal host was throttled more than forty times as much as the TKG cluster despite handling significantly less load. During the periods of highest load, the container on the bare-metal cluster was throttled for about 5 seconds in every 100ms period, while on the TKG cluster it was throttled for about 0.1 second. The high level of throttling on bare metal is possible because the throttled time is the sum of the time that a thread was throttled on any CPU in the system. Five seconds of throttled time is equivalent to 50 threads being throttled across 50 CPUs for the full 100ms. There is some throttling of the container on the TKG cluster, but this is expected due to the use of CPU limits and CPU-intensive nature of the container. On the bare-metal cluster, the large numbers of CPUs in the Kubernetes node combined with the multi-threaded nature of the Java application running in the container caused the kernel to over-throttle the container. To understand why this occurred, it is necessary to understand more about the operation of the CFS bandwidth control algorithm.

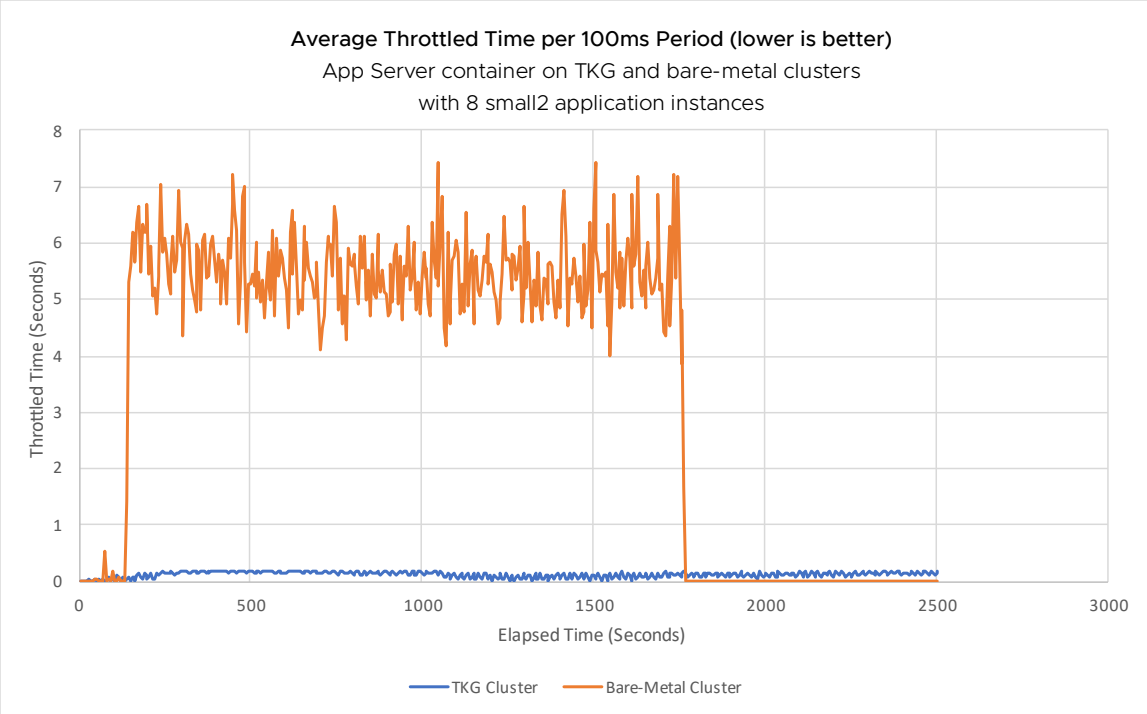

Figure 6. Average throttled time

To enforce the CPU quota, the kernel tracks the amount of remaining quota time in each period. When a thread from the container is scheduled onto a CPU, the kernel allocates a 5ms slice from the quota to that CPU. If the thread does not use the entire slice, then the remaining time, minus 1ms, is eventually returned to the remaining quota. If a thread is scheduled to run on a CPU but no time is left in the quota for the current period, then the thread is not allowed to run. This is true even if another CPU is currently holding an unused slice of the quota.

It is likely that the over-throttling of the Tomcat container on the bare-metal cluster occurred because of this slicing mechanism. On the bare-metal hosts, each SMT thread, of which there are two per core, is seen as an independent CPU. As a result, to the Linux kernel the node contained 256 CPUs. The Tomcat container was running a multi-threaded Java web application, and many of the threads were dedicated to handling short running HTTP requests. These threads could be scheduled across the large number of available CPUs. As soon as a new period started, the 150ms quota was divided into 5ms slices which were allocated to CPUs running threads from the application. However, 150ms / 5ms yields only 30 slices, far fewer than the number of active threads and the number of CPUs on which those threads were scheduled to run. As a result, it is likely that many of the threads were throttled for the entire 100ms period even though there was unused quota time allocated to other CPUs, thus drastically limiting the performance capability of the application on the bare-metal node.

Complete details of the operation of CFS bandwidth control are beyond the scope of this paper, but an excellent explanation can be found here: https://engineering.indeedblog.com/blog/2019/12/cpu-throttling-regression-fix.

It is important to note that far from being a corner case, this over-throttling is likely to be a common issue with multi-threaded applications running on Kubernetes nodes with large numbers of CPUs, such as nodes consisting of a single bare-metal server. There have been many performance issues reported involving Kubernetes limits, some of which may have been caused by this slicing mechanism. Some of these cases are detailed here: https://k8s.af/.

Tuned Bare-Metal Results

To improve the performance of the bare-metal cluster, it was necessary to reduce the impact of throttling due to CPU limits. This was done using the Kubernetes CPU Manager. The CPU Manager is a beta-level feature that enforces CPU limits by allocating exclusive use of a set of CPUs to a pod, rather than using CFS bandwidth control. An overview of the CPU Manager can be found at https://kubernetes.io/blog/2018/07/24/feature-highlight-cpu-manager.

In order to use the CPU Manager for a container within a pod, certain conditions must be met:

- The CPU Manager must be enabled by setting the CPU Manager policy to *static* in the Kubelet settings for the cluster. The CPU Manager is not enabled by default.
- The pod must fall into the Guaranteed QoS class, meaning that requests and limits are set for all containers in the pod, and they are equal.

- The CPU limits for the container must specify a whole number of CPUs.

The first condition was met by enabling the CPU Manager on the bare-metal cluster. The second was already satisfied by the application server pod in the Weathervane small2 configuration. However, the third condition was not met by the small2 configuration, which specified a limit of 1.5 CPUs for the Tomcat container in the application server pod.

To enable the Kubernetes CPU Manager to work with Weathervane, we developed a new configuration in which the CPU limit for the Tomcat container was increased to a whole number of 2 CPUs. In this configuration, called small2-applimit2, the CPU requests for the other pods were increased to handle the higher loads that were enabled by the increased limits. Typically, increasing CPU requests and limits to whole numbers for applications that do not need the increased capacity will result in pods holding resources that are left idle and unavailable for other applications—which might not be ideal. However, as Weathervane is a performance benchmark, the increased limit will be used to drive increased throughput. Figure 7 shows the pods and resources requests and limits for the small2-applimit2 configuration.

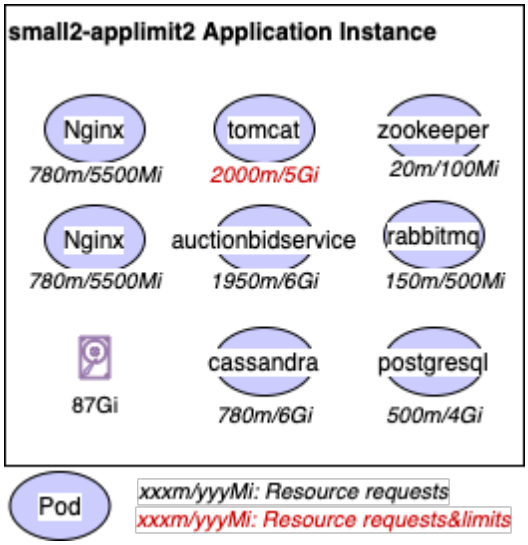

Figure 7. small2-applimit2 configuration

Results

Figure 8 shows a summary of the results obtained by running the Weathervane small2-applimit2 configuration on the TKG cluster and the bare-metal cluster with the CPU Manager enabled. This is the same chart shown in figure 1. The bare-metal performance significantly improved over the original results. However, even with the application and cluster-level tuning, the TKG on vSphere deployment outperformed the bare-metal cluster.

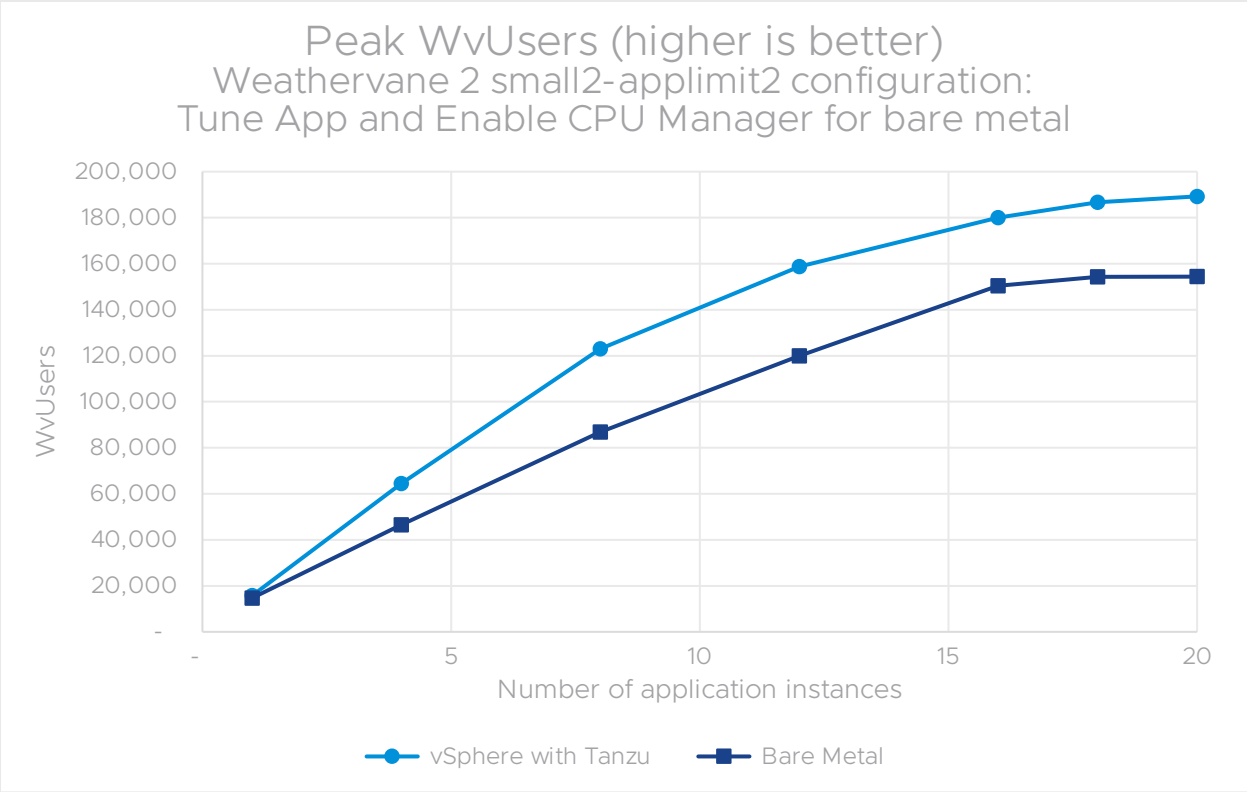

Figure 8. Results with bare-metal CPU Manager enabled and small2-applimit2 configuration size

Discussion

In addition to eliminating the effect of over-throttling due to CPU limits, the bare-metal results in figure 8 reflect the improved compute locality obtained by using the CPU Manager to effectively pin containers to specific CPUs. However, an investigation of the bare-metal results revealed that the CPU Manager was making sub-optimal decisions when assigning the Tomcat containers to physical CPUs, without an apparent config option to alter the behavior. In all cases, the container, which had a limit of 2 CPUs, was assigned to two SMT threads on the same physical core. Because the SMT threads on the same core share resources, this gave a lower performance capability than using two SMT threads on separate cores.

To determine the impact on performance of the placement decisions made by the CPU Manager, we ran tests with SMT disabled on the bare-metal server. The results are shown in figure 9. This is the same chart shown in figure 2. In this case, the performance of the two clusters is essentially the same out to 16 application instances. With SMT disabled, the reduced number of CPUs available for scheduling pods on the bare-metal cluster prevented us from running more than 17 application instances. The moderate CPU over-commitment on the vSphere-with-Tanzu cluster enabled us to run out to 20 application instances. As a result, the peak throughput for the TKG cluster was about 5% higher than for the bare-metal cluster with SMT turned off.

3

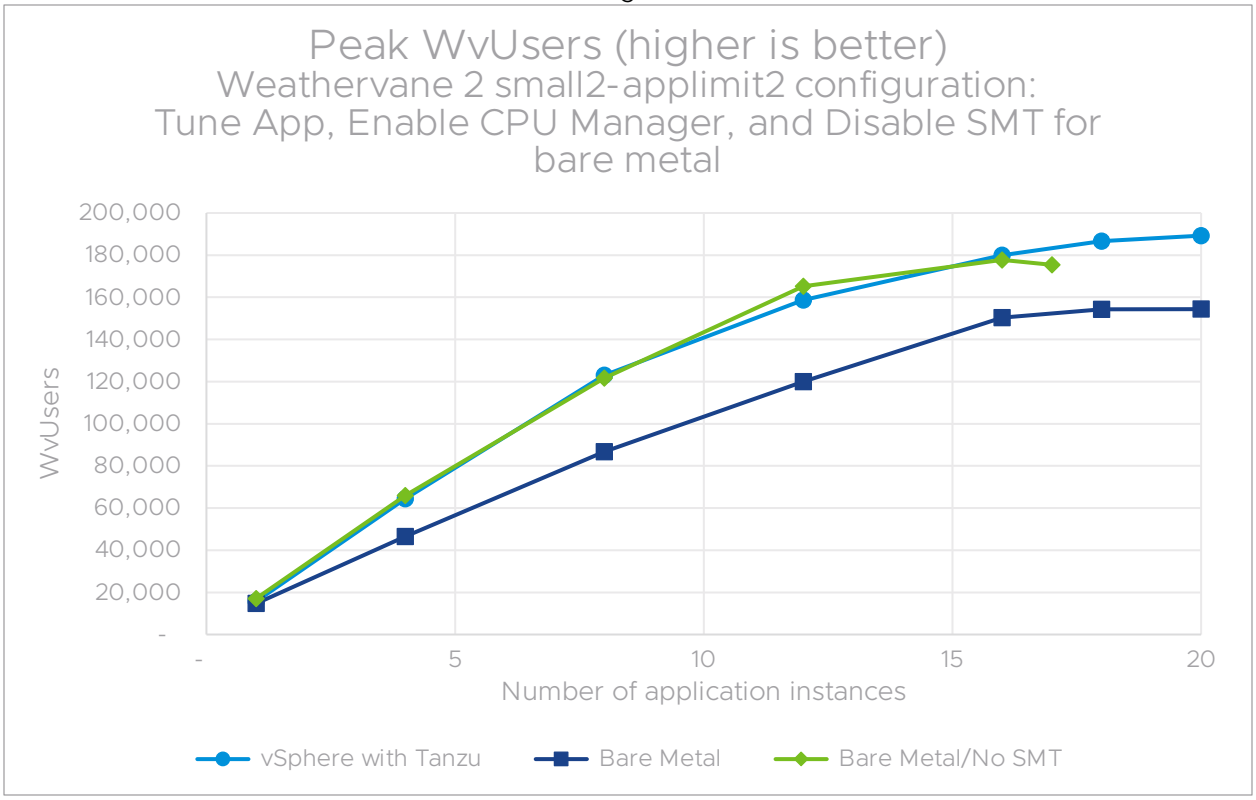

Figure 9. Results with SMT disabled on bare-metal cluster

Conclusion

The results in this paper show that Kubernetes clusters deployed on VMware vSphere with Tanzu can match or exceed the performance of bare-metal clusters deployed on identical servers. In some cases, as shown here when running multi-threaded applications with Kubernetes CPU limits, the vSphere-based clusters can outperform bare metal by a significant margin.

On vSphere, this performance advantage is achieved without the need for significant tuning and configuration steps or the modification of workloads for the use of beta state features like Kubernetes CPU Manager.

As a platform of choice for enterprise applications running on Kubernetes clusters, VMware vSphere with Tanzu provides the benefits of a virtualized infrastructure and achieves excellent performance.

About the Authors

Hal Rosenberg is a VMware alumnus. His focus areas were Kubernetes performance, benchmark development, and performance troubleshooting. He was the architect and lead developer for the Weathervane open-source performance benchmark.

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 developing benchmarks and workloads.