Kubernetes Topology-Aware Scheduling

PUBLIC DOCUMENT

Nov 22, 2019

Authors (add yourself if you change anything):

Chris Friesen (chris.friesen@windriver.com)

Motivation

For the same reasons driving the introduction of additional topology awareness in the pod placement in kubelet itself (5G packet processing, other NFV applications) it's also important to enhance the overall system as a whole in order to make it more likely that a given pod will end up on a node where it will fit. Otherwise you end up with scenarios like https://github.com/kubernetes/kubernetes/issues/84869 where the scheduler keeps making the same bad decision over and over.

Overview

I've been thinking about the work items involved in making the kubernetes scheduler topology-aware so that it's less likely to put a pod on a node where it can't fit. The exact work items would vary depending on the details of the implementation, so I'll cover a few options.

First, there's the question of whether we keep the topology-aware resource tracking within kubelet (like it is now) or whether we publish topology-aware resource availability and usage such that the other kubernetes components (the scheduler, for example) or any other interested party can access it.

Second, there's the question of how we influence the binding of pods to nodes such that pods with topology alignment requirements are more likely to be able to run. Currently the scheduler only looks at per-node resources, not per-NUMA-node, so it's quite easy for it to bind a pod to a node that cannot meet the specified topology affinity. At a high level, there are a number of different options. In order of most to least intrusive:

1. Add a custom scheduler.
2. Enhance the scheduler to be topology-aware.
3. Add a topology-aware controller which will influence the scheduler.

Resource Trackingd

The critical question that the new code needs to answer is whether or not the proposed pod will fit on a given worker node in accordance with the node's Topology Manager policy. (Or in the future, the pod's own specified topology affinity requirements.) This requires knowledge of the proposed pod spec, the available resources on each NUMA node of each worker node, and (currently) the Topology Manager policy of the node. There are essentially two ways of handling this:

1) Per-NUMA-node Resources

Following a similar model as the existing resource tracking, each node would publish per-NUMA-node resources, and would also publish the Topology Manager policy of the node. Other entities in the cluster could then use this information to determine whether a pod can fit on a given node (ideally using the same source code as kubelet to minimize duplication).

The various device plugins can currently report the NUMA-affinity of each device to the device manager in Kubelet, so kubelet would need to be extended to report this information to kube-apiserver and keep it up-to-date. The numerical NUMA node could be specified as a standard suffix (of the form ".<numa_node>" perhaps), or some other syntax could be agreed upon.

The Topology Manager policy could be reported as a node annotation by kubelet.

Going this route has the benefit of consistency and alignment with the existing per-node resources. It also avoids additional overhead at scheduling time as the per-node resource usage data is already available. On the other hand, it would require careful code architecting to ensure that the same code could be re-used between kubelet itself and the higher-level code affecting scheduling. It would also be more susceptible to races as it'll always be using slightly-stale resource data.

2) Call-out to Kubelet

The new scheduler-related code could call out to kubelet to essentially ask it "Would this specified pod fit on you?". This would require adding a new external API call to kubelet, which would be called from the new scheduler-related logic. This would be a new dependency between these components, so we'd have to ensure there are no unforeseen consequences.

The nice thing about this option is that kubelet already knows the Topology Manager policy and already has the logic to check if a pod can fit on the node, though the code might need to be refactored a bit. If the scheduler extension is done via the webhook or sched framework we'd

have to call this for multiple (maybe all) nodes. If we were to modify the kubernetes scheduler proper we could conceivably call it as a final check before binding the pod to the node, which would result in calling it only as many times as needed (but add a delay in case it wouldn't fit). It might make sense to compromise and query in parallel the first N possible nodes in the list of prioritized nodes to see if the pod would fit.

Scheduling Changes

For the scheduler proper, I see a few options for how to handle the additional logic.

1. Create a whole new topology-aware scheduler. One such approach is discussed at https://kccncna19.sched.com/event/UabY/nhd-a-topology-aware-scheduler-for-k8s-for-low-latency-hpc-applications-cliff-burdick-viasat
2. Extend the existing scheduler using a webhook as documented at https://github.com/kubernetes/community/blob/master/contributors/design-proposals/scheduling/scheduler_extender.md I'm pretty sure this would not be able to properly factor in nominated pods in the decision-making.
3. Extend the existing scheduler via a plugin to the scheduler framework as documented at https://github.com/kubernetes/enhancements/blob/master/keps/sig-scheduling/20180409-scheduling-framework.md I think the AddPod()/RemovePod() routines in PreFilterExtensions *might* allow proper consideration of nominated pods, but I'd need to dig into it a bit more (and it wouldn't be trivial). I think there might still be race conditions allowing multiple pods to be simultaneously bound to a node, but this could be mitigated by using the Reserve() plugin API. If we were to use the Kubelet call-out discussed above, it might be possible to implement this as a simplistic check from the Permit() plugin API.
4. A topology-aware higher-level controller could be created to prevent pods from being scheduled on unsuitable nodes, but without scheduler support I think it would be limited to using things like the nodeSelector or nodeAffinity which could get kind of messy and not very granular.

In all four scenarios, we'd use one of the two mechanisms described under "Resource Tracking" above to filter out nodes that would be unable to run the pod according to the topology policy.

In terms of performance, having per-NUMA-node resources published to the cluster would theoretically allow for faster scheduling since it would avoid the round-trip latency of querying kubelet. It might be a bit of a pain to query the per-NUMA-node resources and the Topology Manager policy and convert them into a format that the kubelet code understands though.