The reserve plugin

Tracking pending resources to reduce update pressure

Francesco Romani - fromani@redhat.com - presented 2022/01/10

Rationale

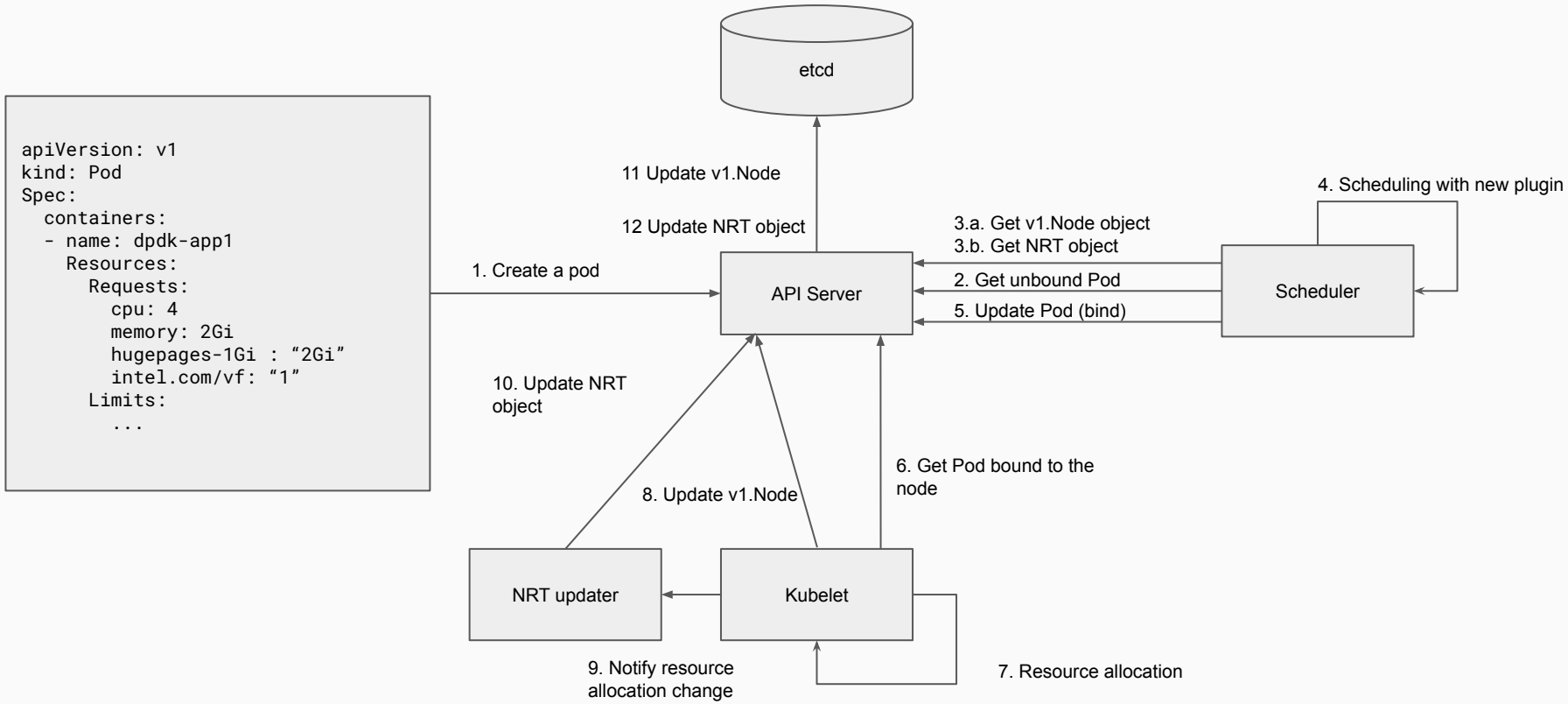
Enhance the noderesourcetopology plugin to **optionally** track pending resources

Pending resources are resources allocated to pods which aren’t yet reported back by the topology updaters running on the worker nodes

Add a new overlay:

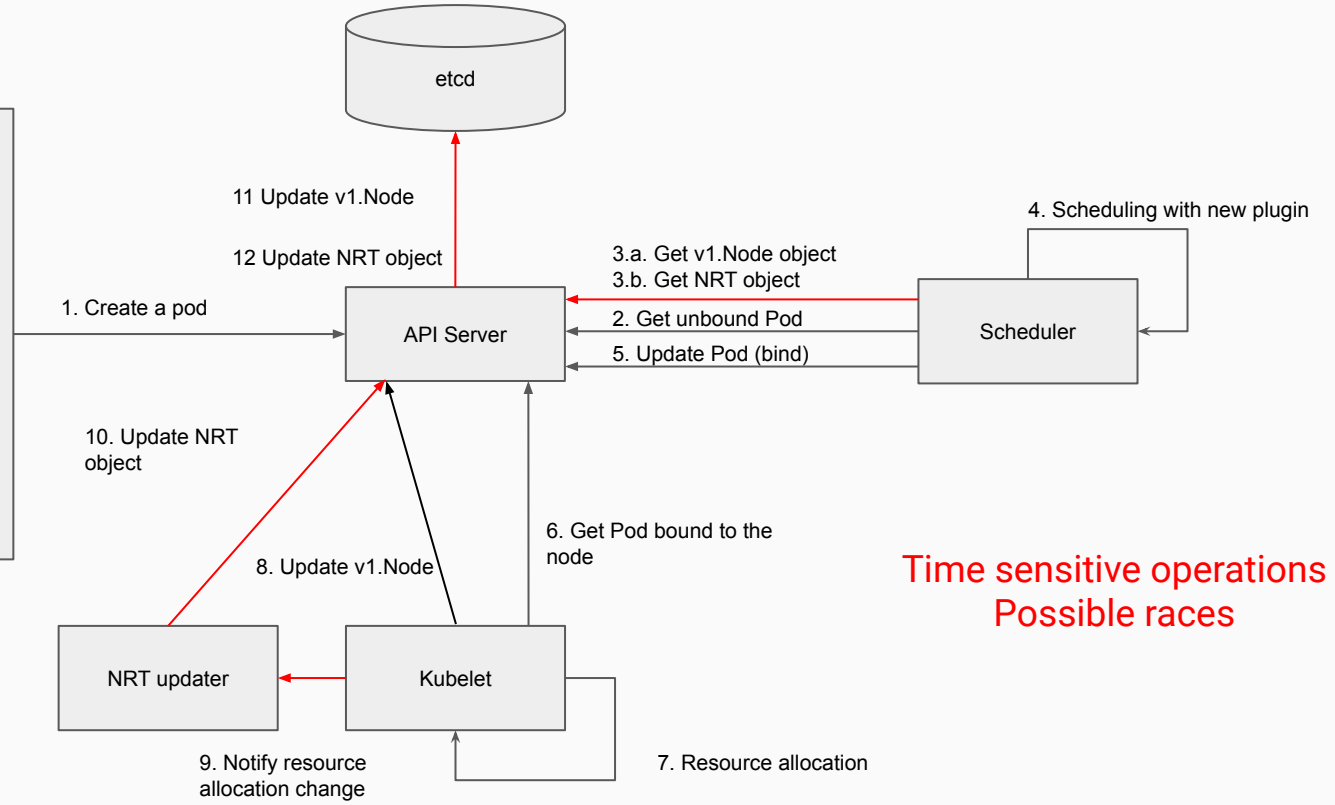
Actual resources = Reported resources - Pending resources

The current architecture (1/2)



The current architecture (2/2)

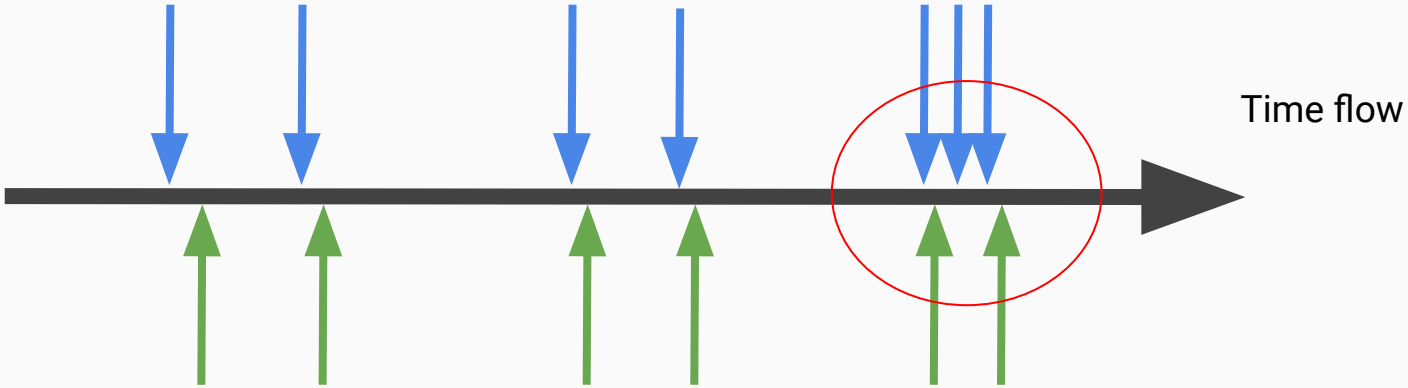
```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - name: dpdk-app1
    Resources:
      Requests:
        cpu: 4
        memory: 2Gi
        hugepages-1Gi : “2Gi”
        intel.com/vf: “1”
      Limits:
        ...
```



Time sensitive operations
Possible races

Data racing: updater v scheduler

Pod to be scheduled - each arrow is a new pod

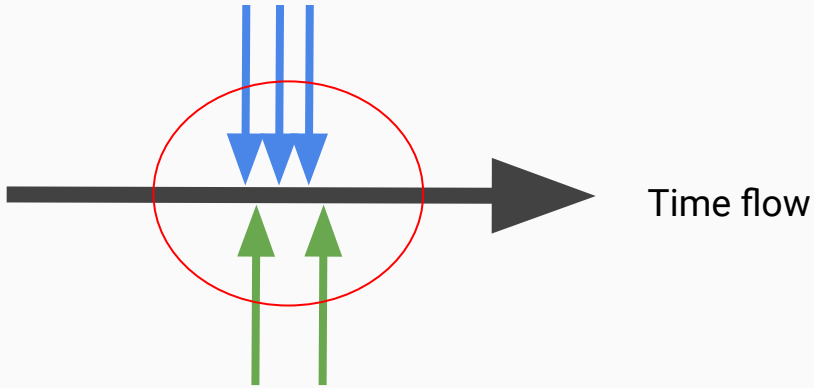


NRT objects updates (as seen by the scheduler) - each arrow is update

Data racing, explained

Without the reserve plugin, or any form of state management:

- The plugin depend on fresh data to make a good scheduling decision
- Regardless of how much the updater is optimized, it is and always will be slower than the scheduler core
- The rate of filling the scheduling pod queue is and will always be faster than the updater rate



What if the scheduler needs to run with stale data?

Fresh data: NRT objects fresher than the last scheduled pod

Stale data: NRT objects OLDER than the last scheduled pod

With Stale Data, the scheduler no longer has a correct picture of the resource allocation in the cluster

Scheduling with stale data (1/2)

In the current framework, scheduling with stale data means

Increase the chance of TopologyAffinityError

Fixing the information imbalance between the scheduler and the kubelet was one of the key goals of the noderesourcestopology plugin

Stale data for the plugin is however another form of information imbalance :(

Information quality = information quantity (enough details) + information freshness (reflects reality)

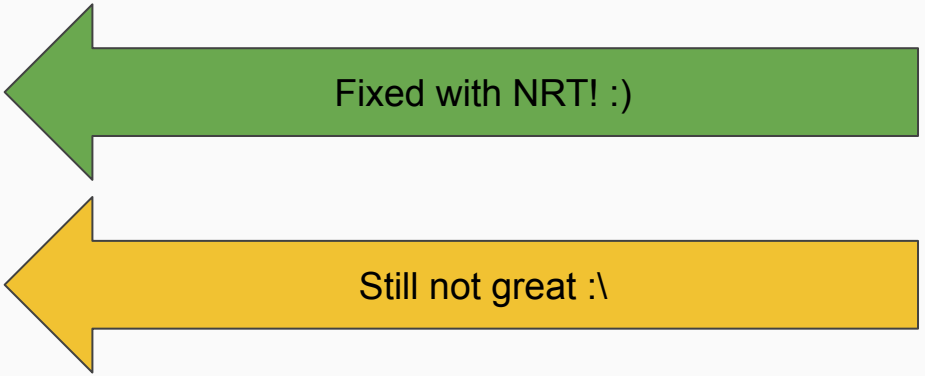
Scheduling with stale data (1/2)

Information quality =

information quantity (enough details)

+

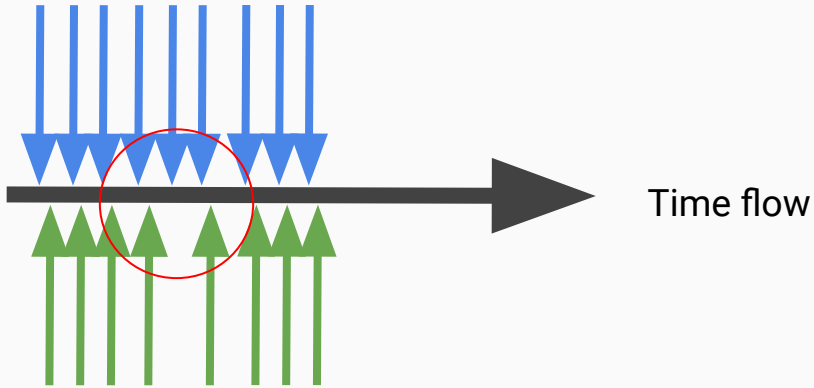
information freshness (reflects reality)



Optimizing the updater

Optimization avenues

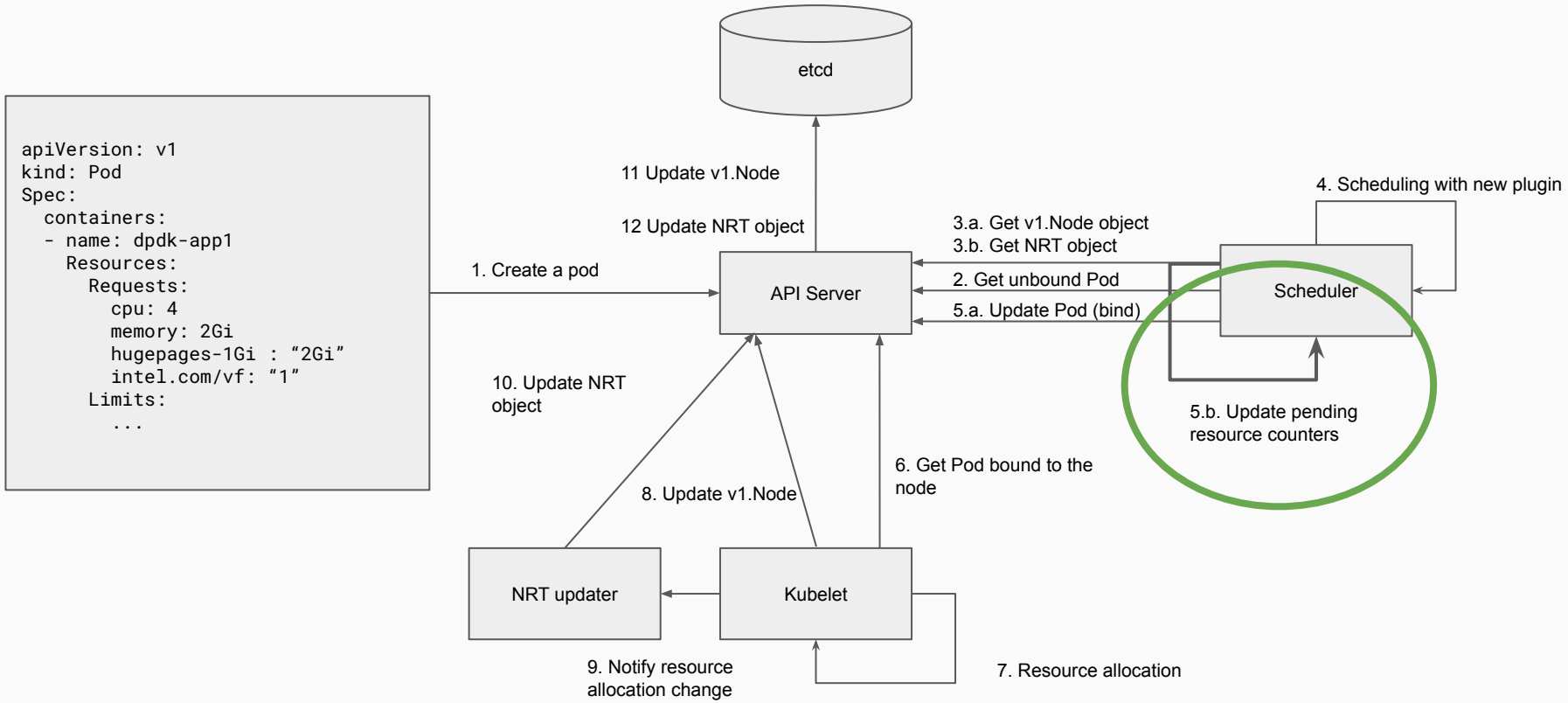
- “Smart polling”:
  - Keep the polling, get notification events to trigger polling in addition to timer
- Watchable podresources endpoints:
  - ListAndWatch
  - GetAllocatableResources AndWatch



Optimizing the updater is only PART of the solution

- No matter how optimized, the updater is intrinsically slower than the scheduler pod queue
  - The updater needs to do MORE steps
  - Upper bound defined by the kubernetes architecture
- The more NRT updates, the more pressure on the APIServer AND Etcd
  - This is another cap to the PRACTICAL update frequency

The proposed architecture

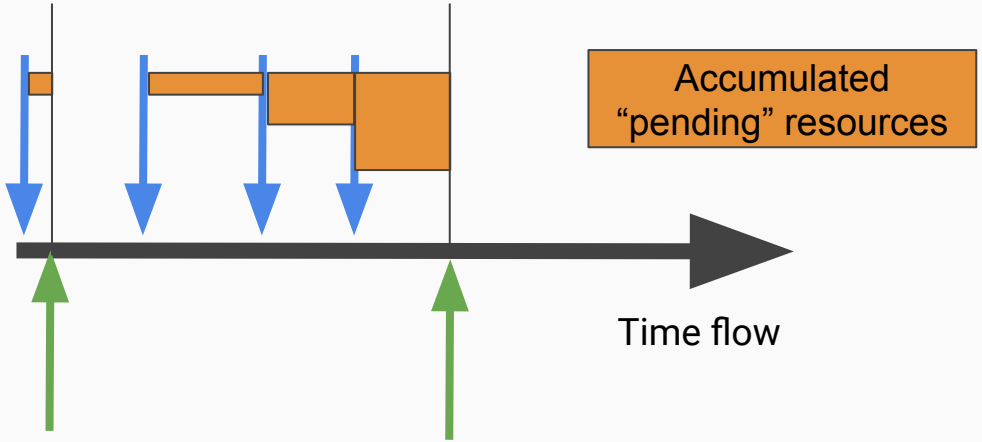


Tracking pending resources

The scheduler knows the resources required by pods!

- Keep a cache of resources allocated but not yet reported by the updater (hence not reflected into NRT objects)
- Invalidate the cache when a new NRT object comes
- Available resources on node = NRT - pending

Works IFF we don't have overlapping schedulers (which we want to avoid in general!)



Available resources = reported - pending

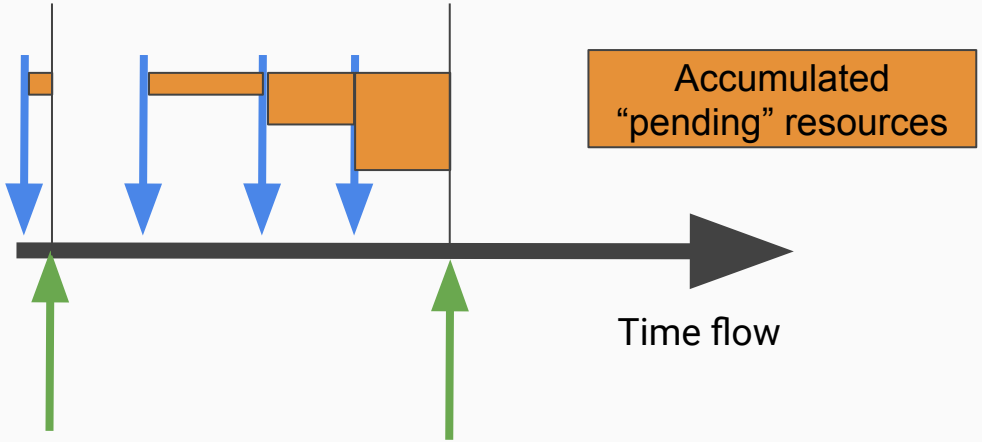
Cache management:

- Accumulate after each successful scheduling (natural fit for reserve plugin)
- Invalidate when fresh data comes
  - Reconcile with real resource availability reported by the updater
- Crash/reboot safe
  - Fits nicely in the client-go behaviour

Caveat: pessimist tracking

The scheduler cannot (and SHOULD NOT) predict on which NUMA node the workload will land

The only option thus is to account the pending resources against ALL the NUMA zones on the given node



Available resources = reported - pending

Consequences of the pessimist tracking:

1. **OVERALLOCATION of pending resources**
2. Nodes may be filtered out
3. If all nodes are filtered out, the pod will be unschedulable
4. Unschedulable pods will be retried later

This is actually a desirable behaviour - as long as is opt-in!

The new tradeoff

Tracking pending resources (the reserve plugin) enables new tradeoffs:

Updating frequency

Versus

worker pool size

Versus

scheduling time

Takeaways for the reserve plugin:

1. Accounting of pending resources is ORTHOGONAL to updater optimization - **we need both!**

2. Accounting of pending resources is OPT-IN: **filter/score plugins will work as before without!**

3. Accounting of pending resources **reduces the pressure** on the updating loop

4. Accounting of pending resources give **more configuration space**: each cluster has a different tradeoff - enable them