kubernetes #71488 issue : #71486

commit 554acf2b38b0d5ba19b62a67c87b2ac0fbd19a9e

问题（原因）：无法满足调度条件(unschedulable)的pod会被收集起来，等待重新分配。节点的更新，运行的pod终止等都会触发重新分配。但是早期的scheduler是不公平的，原因如下：

1. 高优先级的pod因为Affinity或者resource等原因无法被分配，又因为集群事件频繁触发重新调度，导致这些pod反复占用分配队列的队首，造成其余节点饥饿。
2. 假设有许多同样优先级的pod，它们其中一部分是在active queue中，另一部分是在unschedule queue中，那么当在unschedule queue中的pod需要被重新分配时，它们会回到active queue中，由于active queue的数据结构是二项堆，因此那些unschedule queue中的pod会从二项堆的根上加入，从而放置在了那些原本在active queue的pod的前面，造成了它们的饥饿。

#71488针对的是第二种情况

comments in #71486:

In large clusters, events that trigger rescheduling of the unschedulable pods are frequent. For example, in a cluster with 1000 nodes, node updates, which are often just heartbeats, may happen at an average rate of 100/s. These events trigger rescheduling of unschedulable pods. The scheduler moves unschedulable pods to its active scheduling queue when such events are observed. There are two particularly problematic scenarios:

1. There are high-priority unschedulable pods and preemption does not help make them schedulable. For example, they may have node affinity that cannot be satisfied, or they may have pod affinity to other pods that do not exist. Such high priority pods may go to the head of the active scheduling queue and block everything behind them.
2. There are many pending pods with the same priority, but a number of them are unschedulable. When these unschedulable pods are moved to the active queue, they are expected to go behind other pods in the queue, but since our queue sorts pods by priority and these pods have the same priority, their order is unspecified. The Pop() algorithm of the heap swaps the head of the queue with the tail, effectively causing the heap to act similar to a LIFO. So, pods which are tried recently, go to the head with a high probability and are retried. This starves pods behind them in the queue.

复现：无

修复：在active queue二项堆的比较函数中加入了最近被调度的时间，时间越近则位置更靠后。

```
@@ -206,10 +206,31 @@ type PriorityQueue struct {
 // Making sure that PriorityQueue implements SchedulingQueue.
 var _ = SchedulingQueue(&PriorityQueue{})

+// podTimeStamp returns pod's last schedule time or its creation time
if the
```

```
+// scheduler has never tried scheduling it.
+func podTimestamp(pod *v1.Pod) *metav1.Time {
+       _, condition := podutil.GetPodCondition(&pod.Status,
v1.PodScheduled)
+       if condition == nil {
+               return &pod.CreationTimestamp
+       }
+       return &condition.LastTransitionTime
+}
+
+// activeQComp is the function used by the activeQ heap algorithm to
sort pods.
+// It sorts pods based on their priority. When priorities are equal, it
uses
+// podTimestamp.
+func activeQComp(pod1, pod2 interface{}) bool {
+       p1 := pod1.(*v1.Pod)
+       p2 := pod2.(*v1.Pod)
+       prio1 := util.GetPodPriority(p1)
+       prio2 := util.GetPodPriority(p2)
+       return (prio1 > prio2) || (prio1 == prio2 &&
podTimestamp(p1).Before(podTimestamp(p2)))
+}
+
 // NewPriorityQueue creates a PriorityQueue object.
 func NewPriorityQueue() *PriorityQueue {
        pq := &PriorityQueue{
-               activeQ:         newHeap(cache.MetaNamespaceKeyFunc,
util.HigherPriorityPod),
+               activeQ:         newHeap(cache.MetaNamespaceKeyFunc,
activeQComp),
                unschedulableQ: newUnschedulablePodsMap(),
                nominatedPods:  map[string][]*v1.Pod{},
        }
```