kubernetes #53647 issue : #50813, 56261

commit 82e02cc98690096e3d1a43a8613ecf0c6d42656c

问题：当一个node从集群中离开时（如kubectl delete node）， 该node上的pod却没有被删除（1.5版本之前，从node变成unkown状态之后的5分钟k8s会开始清理这些pod，在1.5版本之后，需要用户手动清理）， 这些pod仍然在running状态，当一个新的具有podAntiAffinity的pod进入schedule队列时，会一直陷入pending状态无法被调度。

复现：

How to reproduce it (as minimally and precisely as possible):

- Create a replication controller with a replica of 1;
- Wait until the pod it creates is `Running`;
- Take the node where the pod is running offline;
- Observe that the newly-created pod by the replication controller stays `Pending`.

That said, I wasn't able to repro this on another cluster with the same `kubectl version` and replication controller template. This is not the first time I observe this issue, though.

原因：

在实现上，虽然node已经离开集群，但是只有当pod被删除干净，schedulerCache才会删除掉node，因此该node依然保持在schedulerCache中。运行在该node上的pod会依然存在。

```go
func (cache *schedulerCache) RemoveNode(node *api.Node) error {
	cache.mu.Lock()
	defer cache.mu.Unlock()

	n := cache.nodes[node.Name]
	if err := n.RemoveNode(node); err != nil {
		return err
	}
	// We remove NodeInfo for this node only if there aren't any pods on this node.
	// We can't do it unconditionally, because notifications about pods are delivered
	// in a different watch, and thus can potentially be observed later, even though
	// they happened before node removal.
	if len(n.pods) == 0 && n.node == nil {
		delete(cache.nodes, node.Name)
	}
	return nil
}
```

新的具有PodAntiAffinity的pod进入schedule队列，它会调用CalculateInterPodAffinityPriority函数，该函数会调用GetNodeInfo函数，从而获取ScheduleCache的node信息，从而把已经移

除的node考虑在内，它会返回一个不为空的错误，导致processPod和processNode返回错误，终止CalculateInterPodAffinityPriority，最终导致分配失败，使得该pod处于pending状态。

```
// GetNodeInfo returns cached data for the node 'id'.
func (c *CachedNodeInfo) GetNodeInfo(id string) (*api.Node, error) {
	node, exists, err := c.Get(&api.Node{ObjectMeta:
api.ObjectMeta{Name: id}})

	if err != nil {
		return nil, fmt.Errorf("error retrieving node '%v' from
cache: %v", id, err)
	}

	if !exists {
		return nil, fmt.Errorf("node '%v' not found", id)
	}

	return node.(*api.Node), nil
}
```

修复：在CalculateInterPodAffinityPriority中遍历node上的pod时，首先判断nodeInfo.Node()是否为空，如果为空，则不考虑上面的pod affinity

```
@@ -137,6 +138,10 @@ func (ipa *InterPodAffinity) CalculateInterPodAffinityPriority(pod
*v1.Pod, node
	processPod := func(existingPod *v1.Pod) error {
		existingPodNode, err := ipa.info.GetNodeInfo(existingPod.Spec.NodeName)
		if err != nil {
+			if apierrors.IsNotFound(err) {
+				glog.Errorf("Node not found, %v",
existingPod.Spec.NodeName)
+				return nil
+			}
			return err
		}
		existingPodAffinity := existingPod.Spec.Affinity


@@ -189,19 +194,21 @@ func (ipa *InterPodAffinity) CalculateInterPodAffinityPriority(pod
*v1.Pod, node
	}


	processNode := func(i int) {
		nodeInfo := nodeNameToInfo[allNodeNames[i]]
-		if hasAffinityConstraints || hasAntiAffinityConstraints {
-			// We need to process all the nodes.
-			for _, existingPod := range nodeInfo.Pods() {
-				if err := processPod(existingPod); err != nil {
-					pm.setError(err)
```

```
+               if nodeInfo.Node() != nil {
+                       if hasAffinityConstraints || hasAntiAffinityConstraints {
+                               // We need to process all the nodes.
+                               for _, existingPod := range nodeInfo.Pods() {
+                                       if err := processPod(existingPod); err != nil {
+                                               pm.setError(err)
+                                       }
                                }
-                       }
-               } else {
-                       // The pod doesn't have any constraints - we need to check only
existing
-                       // ones that have some.
-                       for _, existingPod := range nodeInfo.PodsWithAffinity() {
-                               if err := processPod(existingPod); err != nil {
-                                       pm.setError(err)
+                       } else {
+                               // The pod doesn't have any constraints - we need to
check only existing
+                               // ones that have some.
+                               for _, existingPod := range nodeInfo.PodsWithAffinity()
{
+                                       if err := processPod(existingPod); err != nil {
+                                               pm.setError(err)
+                                                }
                                        }
                              }
                    }
```