schedulerframework+Prometheus

使用Prometheus实现scheduler插件扩展

环境版本：cir-containerd:v1.6.24, kubeadm kubeclt kubelet:v1.28.2, Golang1.22.2

使用Prometheus实现根据监控节点网络流量大小对pod进行调度的插件

目录

1. 定义Prometheus结构与方法

为了使用Prometheus监控节点网络流量，需要实现一个prometheusHandle，从prometheus service中获取数据，定义我们需要抓取的数据的时间范围、网络类型等

```
type PrometheusHandle struct {
        deviceName string //需要监控的网络类型（wlan0）
        timeRange time.Duration //抓取数据的时间段
        ip string //prometheus service ip（电院集群：http://10.181.229.225:30278/)
        client promv1.API //操作客户端
}
```

定义prometheusHandle结构体后，需要实现初始化、根据PromQL查询数据、获取数据等函数方法

初始化：

```
//根据ip获取客户端，实例化prometheusHandle结构体
func NewProme(ip, deviceName string, timeRace time.Duration) *PrometheusHandle {
    client, err := api.NewClient(api.Config{Address: ip})
    if err != nil {
        klog.Fatalf("[NetworkTraffic] FatalError creating prometheus client: %s", err.Error())
    }
    return &PrometheusHandle{
        deviceName: deviceName,
        ip:         ip,
        timeRange:  timeRace,
        client:     promv1.NewAPI(client),
    }
}
```

查询:

```
func (p *PrometheusHandle) query(promQL string) (model.Value, error) {
    // 通过promQL查询并返回结果，结果为model.Value类型
    results, warnings, err := p.client.Query(context.Background(), promQL, time.Now())
    if len(warnings) > 0 {
        //报警日志
        klog.Warningf("[NetworkTraffic Plugin] Warnings: %v\n", warnings)
    }
    return results, err
}
```

获取数据:

```
func (p *PrometheusHandle) GetGauge(node string) (*model.Sample, error) {
    //调用查询函数，返回查询到的数据
    value, err := p.query(fmt.Sprintf(nodeMeasureQueryTemplate, p.deviceName, p.timeRange, node))
    fmt.Println(fmt.Sprintf(nodeMeasureQueryTemplate, p.deviceName, p.timeRange, node))
    if err != nil {
        return nil, fmt.Errorf("[NetworkTraffic] Error querying prometheus: %w", err)
    }
    //未查到数据时报错
    nodeMeasure := value.(model.Vector)
    if len(nodeMeasure) != 1 {
        return nil, fmt.Errorf("[NetworkTraffic] Invalid response, expected 1 value, got %d", len(nodeMeasure))
    }
    return nodeMeasure[0], nil
}
```

为了从配置文件中读取我们需要的参数，还需定义如下参数结构，满足KubeSchedulerConfiguration可以解析的条件：

```
type NetworkTrafficArgs struct {
    IP         string `json:"ip"`
    DeviceName string `json:"deviceName"`
    TimeRange  int    `json:"timeRange"`
}
```

实例化插件时，使用framework.DecodeInto扩展该资源类型：

```
func createNew(ctx context.Context, plArgs runtime.Object, f framework.Handle)
(framework.Plugin, error) {
    args := &NetworkTrafficArgs{}
    if err := frameworkruntime.DecodeInto(plArgs, args); err != nil {
        return nil, err
    }

    klog.Infof("[NetworkTraffic] args received. Device: %s; TimeRange: %d,
Address: %s", args.DeviceName, args.TimeRange, args.IP)

    return &NetworkTraffic{
        handle: f,
        prometheus: NewProme(args.IP, args.DeviceName,
time.Second*time.Duration(args.TimeRange)),
    }, nil
}
```

为了查询节点网络流量，PromQL为:

```
//一对多的查询节点的网络流量，参考PromQL文档
const (
    nodeMeasureQueryTemplate =
"sum_over_time(node_network_receive_bytes_total{device=\"%s\"}[%s]) *
on(instance) group_left(nodename) (node_uname_info{instance=\"%s\"})"
)
```

2. 实现调度插件逻辑

选用Score扩展点，大致逻辑为网络流量越高，节点的score越低，优先将pod分配到网络流量低的节点上

同时需要注意，在实现Score扩展点的同时，还需要实现NormalizeScore

```
//定义插件API与结构体
const Name = "NetworkTraffic"
var _ framework.ScorePlugin = &NetworkTraffic{}

type NetworkTraffic struct {
    prometheus *PrometheusHandle
    handle framework.Handle
}
func (n *NetworkTraffic) Name() string {
        return Name
}
//实现Score，调用prometheusHandle的GetGauge方法，获取节点的网络流量
func (n *NetworkTraffic) Score(ctx context.Context, state
*framework.CycleState, p *corev1.Pod, nodeName string) (int64,
*framework.Status) {
        nodeBandwidth, err := n.prometheus.GetGauge(nodeName)
        if err != nil {
                return 0, framework.NewStatus(framework.Error,
fmt.Sprintf("error getting node bandwidth: %s",err))
        }
        bandWidth := int64(nodeBandwidth.Value)
        klog.Infof("[NetworkTraffic] node '%s' bandwidth: %v",nodeName,
bandWidth)
        return bandWidth, framework.NewStatus(framework.Success, "")
}
func (n *NetworkTraffic) ScoreExtensions() framework.ScoreExtensions {
    return n
}

//实现NormalizeScore，根据网络流量给节点赋值，并保证Score小于MaxNodeScore
func (n *NetworkTraffic) NormalizeScore(ctx context.Context, state
*framework.CycleState, pod *corev1.Pod, scores framework.NodeScoreList)
*framework.Status {
        var higherScore int64 = 0
        for _, node := range scores{
                if higherScore < node.Score {
                        higherScore = node.Score
                }
        }

        klog.Infof("[NetworkTraffic] highest score: %v", higherScore)
        for i, node := range scores {
                klog.Infof("[NetworkTraffic] operation: %v - ( %v * 100 /
%v)", framework.MaxNodeScore ,node.Score, higherScore)
                scores[i].Score = framework.MaxNodeScore - (node.Score * 100 /
higherScore)
                klog.Infof("[NetworkTraffic] node '%s' final score: %v", node
,scores[i].Score)
        }

        klog.Infof("[NetworkTraffic] Nodes final score: %v", scores)
    return framework.NewStatus(framework.Success, "")
}
```

最后补充上main函数入口，同时import需要的包（注意需要导入prometheus需要的一些依赖包）

```
package main

import (
        "context"
        "fmt"
        "os"
        "time"

        corev1 "k8s.io/api/core/v1"
        "k8s.io/apimachinery/pkg/runtime"
        "k8s.io/component-base/logs"
        "k8s.io/klog/v2"
        "github.com/prometheus/client_golang/api"
        promv1 "github.com/prometheus/client_golang/api/prometheus/v1"
        "github.com/prometheus/common/model"
        "k8s.io/kubernetes/cmd/kube-scheduler/app"
        "k8s.io/kubernetes/pkg/scheduler/framework"
        frameworkruntime "k8s.io/kubernetes/pkg/scheduler/framework/runtime"
)

func main() {
        // rand.Seed(time.Now().UTC().UnixNano())

        command := app.NewSchedulerCommand(
                app.WithPlugin(Name, createNew),
        )

        logs.InitLogs()
        defer logs.FlushLogs()
        if err := command.Execute(); err != nil {
                _, _ = fmt.Fprintf(os.Stderr, "%v\n", err)
                os.Exit(1)
        }

}
```

3. 部署项目

Dockerfile如下（将CMD行修改为我们的插件名称）：

```
FROM busybox:stable-musl
WORKDIR /bin
ADD . .
RUN chmod 777  networkTraffic
CMD ["networkTraffic","--v=3","--config=/etc/kubernetes/scheduler-config.yaml"]
```

使用role.yaml来部署调度器（修改ConfigMap与镜像地址）：

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sample-scheduler-clusterrole
rules:
  - apiGroups:
      - ""
    resources:
      - endpoints
      - events
    verbs:
      - create
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - delete
      - get
      - list
      - watch
      - update
  - apiGroups:
      - ""
    resources:
      - bindings
      - pods/binding
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - pods/status
    verbs:
      - patch
      - update
  - apiGroups:
      - ""
    resources:
      - replicationcontrollers
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - apps
      - extensions
    resources:
      - replicasets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - apps
    resources:
      - statefulsets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - policy
    resources:
      - poddisruptionbudgets
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - persistentvolumeclaims
      - persistentvolumes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "storage.k8s.io"
    resources:
      - storageclasses
      - csinodes
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "coordination.k8s.io"
    resources:
      - leases
    verbs:
      - create
      - get
      - list
      - update
```

```yaml
  - apiGroups:
      - "events.k8s.io"
    resources:
      - events
    verbs:
      - create
      - patch
      - update
  - apiGroups:
    - "storage.k8s.io"
    resources:
      - csistoragecapacities
      - csidrivers
    verbs:
      - get
      - list
      - watch
  - apiGroups:
    - ""
    resources:
      - namespaces
    verbs:
      - list
      - watch

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sample-scheduler-sa
  namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: sample-scheduler-clusterrolebinding
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: sample-scheduler-clusterrole
subjects:
  - kind: ServiceAccount
    name: sample-scheduler-sa
    namespace: kube-system
---
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: scheduler-config
  namespace: kube-system
data:
  scheduler-config.yaml: |
    apiVersion: kubescheduler.config.k8s.io/v1
    kind: KubeSchedulerConfiguration
    leaderElection:
      leaderElect: false
    profiles:
    - schedulerName: sample-scheduler
      plugins:
        score:
          enabled:
          - name: "NetworkTraffic"
          disabled:
          - name: "*"
      pluginConfig:
        - name: "NetworkTraffic"
          args:
            ip: "http://10.181.229.225:30278" #替换为集群的promethus serviceIP
            deviceName: "wlan0" #需要监测的网络类型
            timeRange: 15 #监测数据的时间段（单位:s）
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-scheduler
  namespace: kube-system
  labels:
    component: sample-scheduler
spec:
  replicas: 1
  selector:
    matchLabels:
      component: sample-scheduler
  template:
    metadata:
      labels:
        component: sample-scheduler
    spec:
      serviceAccount: sample-scheduler-sa
      priorityClassName: system-cluster-critical
      volumes:
        - name: scheduler-config
          configMap:
            name: scheduler-config
      containers:
        - name: scheduler-ctrl
          image: registry.cn-hangzhou.aliyuncs.com/my_k8s_learning/temp_scheduler:networkscheduler0.1.5 #镜像地址
          imagePullPolicy: IfNotPresent
          args:
            - --config=/etc/kubernetes/scheduler-config.yaml
            - --v=3
          resources:
            requests:
              cpu: "50m"
          volumeMounts:
            - name: scheduler-config
              mountPath: /etc/kubernetes
          command: ["networkTraffic"]
```

使用test.yaml测试调度器：

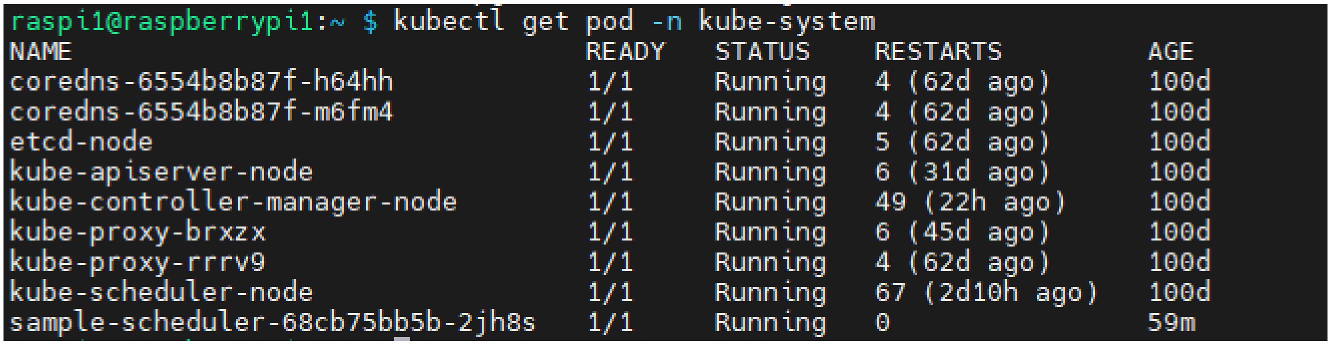
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-scheduler
spec:
  replicas: 5
  selector:
    matchLabels:
      app: test-scheduler
  template:
    metadata:
      labels:
        app: test-scheduler
    spec:
      schedulerName: sample-scheduler
      containers:
        - image: registry.cn-hangzhou.aliyuncs.com/temp-iiip/temp:busybox
          name: busybox-try
          command: ["sleep"]
          args: ["infinity"]
```

4. 结果验证

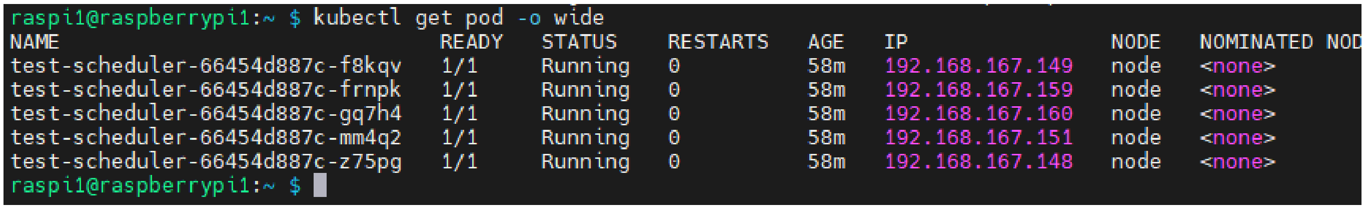
运行指令部署调度器和测试Pod：

```
kubectl create -f /home/role.yaml
kubectl create -f /home/test.yaml
```

可以看到sample-scheduler成功部署

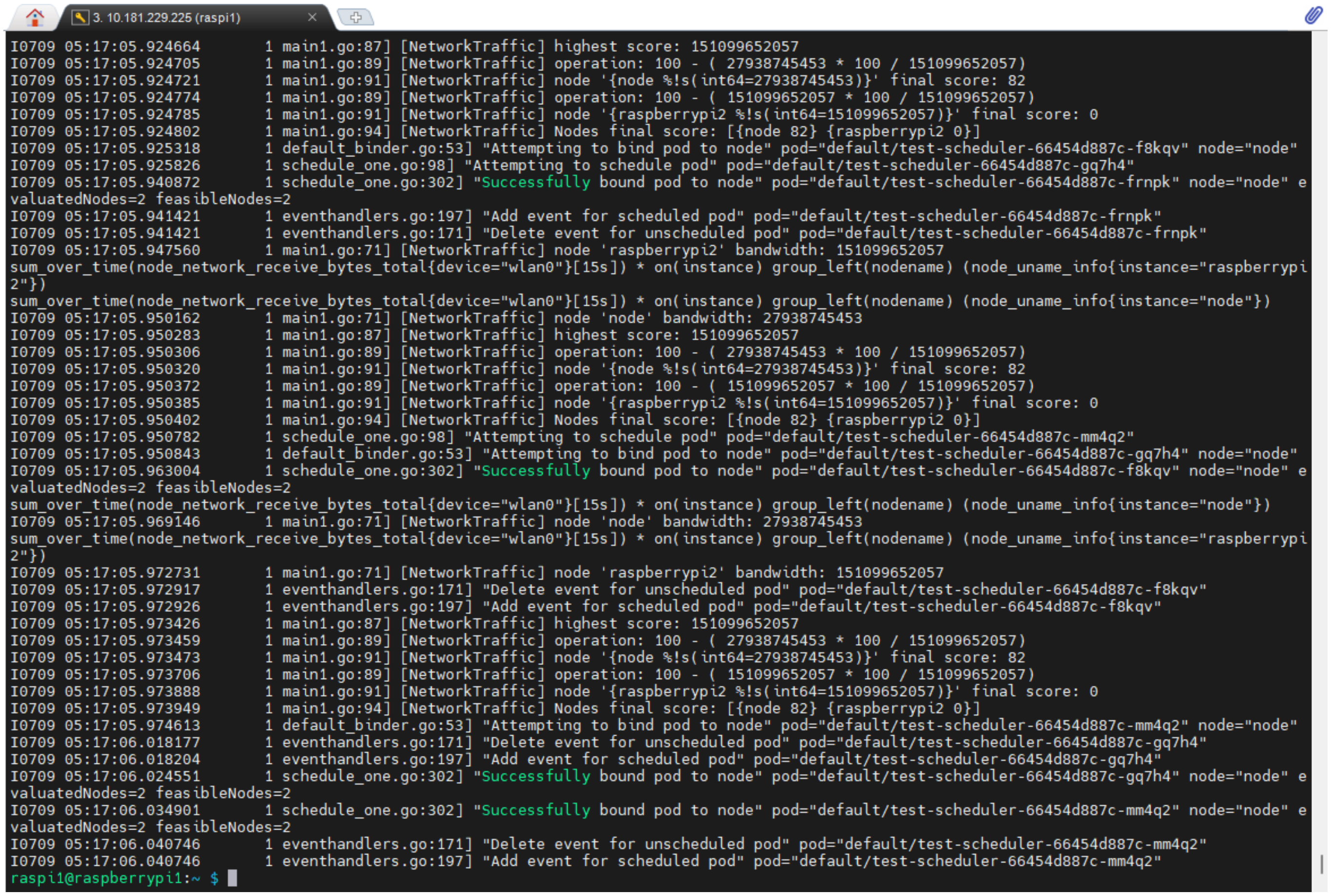


```
raspi1@raspberrypi1:~ $ kubectl get pod -n kube-system
NAME                                READY   STATUS    RESTARTS         AGE
coredns-6554b8b87f-h64hh            1/1     Running   4 (62d ago)      100d
coredns-6554b8b87f-m6fm4            1/1     Running   4 (62d ago)      100d
etcd-node                           1/1     Running   5 (62d ago)      100d
kube-apiserver-node                 1/1     Running   6 (31d ago)      100d
kube-controller-manager-node        1/1     Running   49 (22h ago)     100d
kube-proxy-brxzx                    1/1     Running   6 (45d ago)      100d
kube-proxy-rrrv9                    1/1     Running   4 (62d ago)      100d
kube-scheduler-node                 1/1     Running   67 (2d10h ago)   100d
sample-scheduler-68cb75bb5b-2jh8s   1/1     Running   0                59m
```



```
raspi1@raspberrypi1:~ $ kubectl get pod -o wide
NAME                               READY   STATUS    RESTARTS   AGE   IP                NODE   NOMINATED NOD
test-scheduler-66454d887c-f8kqv    1/1     Running   0          58m   192.168.167.149   node   <none>
test-scheduler-66454d887c-frnpk    1/1     Running   0          58m   192.168.167.159   node   <none>
test-scheduler-66454d887c-gq7h4    1/1     Running   0          58m   192.168.167.160   node   <none>
test-scheduler-66454d887c-mm4q2    1/1     Running   0          58m   192.168.167.151   node   <none>
test-scheduler-66454d887c-z75pg    1/1     Running   0          58m   192.168.167.148   node   <none>
raspi1@raspberrypi1:~ $ 
```

查询sample-scheduler日志：



```
I0709 05:17:05.924664       1 main1.go:87] [NetworkTraffic] highest score: 151099652057
I0709 05:17:05.924705       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 27938745453 * 100 / 151099652057)
I0709 05:17:05.924721       1 main1.go:91] [NetworkTraffic] node '{node %!s(int64=27938745453)}' final score: 82
I0709 05:17:05.924774       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 151099652057 * 100 / 151099652057)
I0709 05:17:05.924785       1 main1.go:91] [NetworkTraffic] node '{raspberrypi2 %!s(int64=151099652057)}' final score: 0
I0709 05:17:05.924802       1 main1.go:94] [NetworkTraffic] Nodes final score: [{node 82} {raspberrypi2 0}]
I0709 05:17:05.925318       1 default_binder.go:53] "Attempting to bind pod to node" pod="default/test-scheduler-66454d887c-f8kqv" node="node"
I0709 05:17:05.925826       1 schedule_one.go:98] "Attempting to schedule pod" pod="default/test-scheduler-66454d887c-gq7h4"
I0709 05:17:05.940872       1 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-frnpk" node="node" e
valuatedNodes=2 feasibleNodes=2
I0709 05:17:05.941421       1 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-frnpk"
I0709 05:17:05.941421       1 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-frnpk"
I0709 05:17:05.947560       1 main1.go:71] [NetworkTraffic] node 'raspberrypi2' bandwidth: 151099652057
sum_over_time(node_network_receive_bytes_total{device="wlan0"}[15s]) * on(instance) group_left(nodename) (node_uname_info{instance="raspberrypi
2"})
sum_over_time(node_network_receive_bytes_total{device="wlan0"}[15s]) * on(instance) group_left(nodename) (node_uname_info{instance="node"})
I0709 05:17:05.950162       1 main1.go:71] [NetworkTraffic] node 'node' bandwidth: 27938745453
I0709 05:17:05.950283       1 main1.go:87] [NetworkTraffic] highest score: 151099652057
I0709 05:17:05.950306       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 27938745453 * 100 / 151099652057)
I0709 05:17:05.950320       1 main1.go:91] [NetworkTraffic] node '{node %!s(int64=27938745453)}' final score: 82
I0709 05:17:05.950372       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 151099652057 * 100 / 151099652057)
I0709 05:17:05.950385       1 main1.go:91] [NetworkTraffic] node '{raspberrypi2 %!s(int64=151099652057)}' final score: 0
I0709 05:17:05.950402       1 main1.go:94] [NetworkTraffic] Nodes final score: [{node 82} {raspberrypi2 0}]
I0709 05:17:05.950782       1 schedule_one.go:98] "Attempting to schedule pod" pod="default/test-scheduler-66454d887c-mm4q2"
I0709 05:17:05.950843       1 default_binder.go:53] "Attempting to bind pod to node" pod="default/test-scheduler-66454d887c-gq7h4" node="node"
I0709 05:17:05.963004       1 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-f8kqv" node="node" e
valuatedNodes=2 feasibleNodes=2
sum_over_time(node_network_receive_bytes_total{device="wlan0"}[15s]) * on(instance) group_left(nodename) (node_uname_info{instance="node"})
I0709 05:17:05.969146       1 main1.go:71] [NetworkTraffic] node 'node' bandwidth: 27938745453
sum_over_time(node_network_receive_bytes_total{device="wlan0"}[15s]) * on(instance) group_left(nodename) (node_uname_info{instance="raspberrypi
2"})
I0709 05:17:05.972731       1 main1.go:71] [NetworkTraffic] node 'raspberrypi2' bandwidth: 151099652057
I0709 05:17:05.972917       1 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-f8kqv"
I0709 05:17:05.972926       1 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-f8kqv"
I0709 05:17:05.973426       1 main1.go:87] [NetworkTraffic] highest score: 151099652057
I0709 05:17:05.973459       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 27938745453 * 100 / 151099652057)
I0709 05:17:05.973473       1 main1.go:91] [NetworkTraffic] node '{node %!s(int64=27938745453)}' final score: 82
I0709 05:17:05.973706       1 main1.go:89] [NetworkTraffic] operation: 100 - ( 151099652057 * 100 / 151099652057)
I0709 05:17:05.973888       1 main1.go:91] [NetworkTraffic] node '{raspberrypi2 %!s(int64=151099652057)}' final score: 0
I0709 05:17:05.973949       1 main1.go:94] [NetworkTraffic] Nodes final score: [{node 82} {raspberrypi2 0}]
I0709 05:17:05.974613       1 default_binder.go:53] "Attempting to bind pod to node" pod="default/test-scheduler-66454d887c-mm4q2" node="node"
I0709 05:17:06.018177       1 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-gq7h4"
I0709 05:17:06.018204       1 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-gq7h4"
I0709 05:17:06.024551       1 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-gq7h4" node="node" e
valuatedNodes=2 feasibleNodes=2
I0709 05:17:06.034901       1 schedule_one.go:302] "Successfully bound pod to node" pod="default/test-scheduler-66454d887c-mm4q2" node="node" e
valuatedNodes=2 feasibleNodes=2
I0709 05:17:06.040746       1 eventhandlers.go:171] "Delete event for unscheduled pod" pod="default/test-scheduler-66454d887c-mm4q2"
I0709 05:17:06.040746       1 eventhandlers.go:197] "Add event for scheduled pod" pod="default/test-scheduler-66454d887c-mm4q2"
raspi1@raspberrypi1:~ $ 
```

可以看到Pod成功绑定到对应的节点。