# 一 运行采集程序

采集脚本采集服务器功率、cpu频率等信息，并进行存储。
运行方式： nohup python ./base_code/set_power_and_cpu_usage.py &

# 二 开关机

## 调度

1. 命令格式
   python ./open_close_code/dispatch/open_machine.py
   python ./open_close_code/dispatch/close_machine.py
2. 命令解释
   以上两个python脚本分别对应开关机的能效优化
   开机逻辑：遍历所有的pod，当pod调度失败的原因是因为资源不足时，我们开启节点，给出要开启节点的名称。
   关机逻辑：遍历所有的节点，当节点上没有pod运行时，关闭节点，给出要关闭节点的名称。
3. 返回结果格式

```
{ “powerOn”:[ "node14","node15" ] }
```

```
{ “powerOff”:[ "node14","node15" ] }
```

1. 示例
   将结果写到了指定文件，关机的写到了/workspace/schedule/power/mac_close.json；开机的写到了/workspace/schedule/power/mac_open.json
   {"powerOn": ["k8s-master"]}

{"powerOff": ["k8s-node"]}

## 迁移

1. 命令格式
   python ./open_close_code/migrate/migrate.py
2. 命令解释
   以上python脚本对应开关机的能效优化 :寻找集群中运行pod个数最少的num个节点（配置文件中定义），将这num个节点上运行的pod迁移到其他机器上，并将这些机器关闭。返回结果：需要迁移的pod和目标节点以及需要关闭的节点。
3. 返回结果格式

```
{ "pods": [ { "podName":"namexx1", "nameSpace":"default", "nodeName":"node14" }, {}, { "podName":"namexx2", "nameSpace":"default", "nodeName":"node15" } ]
```

4. 示例
将结果写到了指定文件，/root/workspace/schedule/power/close_machine_migrate.json




# 三 DVFS（基于使用率频率的优化）

## 调度

1. 命令格式
   python ./dvfs_code/dispatch/adjust_freq.py
   python ./dvfs_code/dispatch/set_freq.py
   sudo ./dvfs_code/dispatch/dvfsScheduler/dvfsScheduler 提前编译好基于extender的调度程序dvfsScheduler，配置k8s调度器，配置完成后，在调度时即可调用dvfsScheduler程序
2. 命令解释
   第一二条命令（两个独立程序，一个负责监控一个负责调节）：实时监控CPU利用率，当某个机器的利用率低于某个数值时，通过dvfs把该机的cpu频率降低。当CPU利用率上升时，再通过dvfs恢复cpu频率。
   第三条命令：完成extender的运行，k8s在筛选节点时，只在没有其它可用节点的情况下，才考虑将pod调度在以最低频率运行的机器，如果选中了某台低频率机器，先把它的频率恢复正常。
3. 返回结果格式
   无需开关节点，无需进行迁移，由k8s调度器以及extender共同完成调度，我们的python脚本完成频率调节。

4. 执行结果

```
cpu利用率低，进行频率调节，将频率调小
降低机器频率结果为：powersave

[redacted]

cpu利用率升高，进行频率调节，将频率调大
升高机器频率结果为：performance
```

```
root@k8s-master:/usr/local/go/myscheduler/energy-efficiency-aware-scheduler-main/test# python3 test.py
当前cpu状态为： performance
当前需要设置的cpu状态为： powersave
调整服务器状态为powersave
tee: '/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor': No such file or directory
降低机器频率结果为：powersave

node/k8s-node not labeled

当前cpu状态为： performance
当前需要设置的cpu状态为： powersave
调整服务器状态为powersave
tee: '/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor': No such file or directory
降低机器频率结果为：powersave
```

```
2023/06/09 21:51:56 --------+++++++++filter++++++++++++------------------
2023/06/09 21:51:56 pod pi-6rqnj/default is lucky to fit on node k8s-master
2023/06/09 21:51:56 node:k8s-master通过筛选
2023/06/09 21:51:56 pod pi-6rqnj/default is lucky to fit on node k8s-node
2023/06/09 21:51:56 node:k8s-node通过筛选
2023/06/09 21:51:56 --------+++++++++prioritize++++++++++++------------------
2023/06/09 21:51:56 节点k8s-master的cpu状态为:performance
2023/06/09 21:51:56 pod pi-6rqnj/default is lucky to get score 10 on node k8s-master
2023/06/09 21:51:56 节点k8s-node的cpu状态为:powersave
2023/06/09 21:51:56 pod pi-6rqnj/default is lucky to get score 0 on node k8s-node
2023/06/09 21:51:56
```

## 迁移

1. 命令格式

   python ./dvfs_code/migrate/dvfs_migrate.py
2. 命令解释

   设置几台空闲节点，实时监控CPU利用率，当某个机器的利用率高于某个数值时，把该节点上的pod迁移到利用率低的节点上，直到低于某个阈值。
3. 返回结果格式

```
{ "pods": [ { "podName":"namexx1", "nameSpace":"default", "nodeName":"node14" }, {}, { "podName":"namexx2", "nameSpace":"default", "nodeName":"node15" } ]
```

1. 执行结果

```
{"pods": [{"podName": "pi-fkpmh", "nameSpace": "default", "nodeName": "k8s-master"}]}
```

```
root@k8s-master:/usr/local/go/myscheduler/energy-efficiency-aware-scheduler-main/test# python3 test.py
当前节点的cpu状态为： performance
当前节点的cpu利用率为： 80
k8s-master的状态为： powersave
k8s-node的状态为： performance
以下节点的状态为powersave，可以向上调度任务
k8s-master
k8s-node 节点上的pod pi-mctx8需要被迁移走，应该迁移到k8s-master 上
root@k8s-master:/usr/local/go/myscheduler/energy-efficiency-aware-scheduler-main/test# python3 test.py
当前节点的cpu状态为： performance
当前节点的cpu利用率为： 47
cpu利用率足够小，停止迁移
tee: '/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor': No such file or directory
降低机器频率结果为：powersave

root@k8s-master:/usr/local/go/myscheduler/energy-efficiency-aware-scheduler-main/test# python3 test.py
当前节点的cpu状态为： powersave
当前节点的cpu利用率为： 47
当前节点为关闭节点，无需迁移
root@k8s-master:/usr/local/go/myscheduler/energy-efficiency-aware-scheduler-main/test#
```

# 四 power capping（基于功率的优化）

## 调度

1. 命令格式

   python ./power_capping_code/dispatch/adjust_power.py

   python ./power_capping_code/dispatch/set_capping.py

sudo ./power_capping_code/dispatch/powerCappingScheduler/powerCappingScheduler 提前编译好基于extender的调度程序powerCappingScheduler，配置k8s调度器，配置完成后，在调度时即可调用powerCappingScheduler程序

2. 命令解释
   第一、二条命令（两个独立程序，一个负责监控一个负责调节）：实时监控服务器功率，当某个机器的功率高于最大功率的一个百分比时，通过power capping限制该机的功率。
   第三条命令：完成extender的运行，k8s在筛选节点时，只在没有其它可用节点的情况下，才考虑将pod调度在被限制功率的机器上，如果选中了某台被限制功率的机器，先解除功率限制。
3. 返回结果格式
   无需开关节点，无需进行迁移，由k8s调度器以及extender共同完成调度，我们的python脚本完成功率限制/取消功率限制。
4. 执行结果

```
当前capping状态为： uncapping
node/k8s-node not labeled

当前capping状态为： uncapping
服务器实时功率过高，进行power capping操作
capping操作成功！
node/k8s-node labeled

当前capping状态为： capping
node/k8s-node not labeled
```

```
当前capping状态为： capping
当前需要设置的capping状态为： uncapping
调整服务器状态为uncapping
uncapping操作成功！
node/k8s-node labeled

当前capping状态为： uncapping
当前需要设置的capping状态为： uncapping
node/k8s-node not labeled

当前capping状态为： uncapping
当前需要设置的capping状态为： uncapping
node/k8s-node not labeled
```

```
2023/06/13 23:52:15 --------+++++++++filter++++++++++++-------------------
2023/06/13 23:52:15 pod pi-4czdc/default is lucky to fit on node k8s-master
2023/06/13 23:52:15 node:k8s-master通过筛选
2023/06/13 23:52:15 pod pi-4czdc/default is lucky to fit on node k8s-node
2023/06/13 23:52:15 node:k8s-node通过筛选
2023/06/13 23:52:15 --------+++++++++prioritize++++++++++++-------------------
2023/06/13 23:52:15 节点k8s-master的cpu状态为:uncapping
2023/06/13 23:52:15 pod pi-4czdc/default is lucky to get score 10 on node k8s-master
2023/06/13 23:52:15 节点k8s-node的cpu状态为:capping
2023/06/13 23:52:15 pod pi-4czdc/default is lucky to get score 0 on node k8s-node
2023/06/13 23:52:15
2023/06/13 23:52:15 --------++++++++++filter++++++++++++-------------------
```

# 迁移

1. 命令格式
   python ./power_capping_code/migrate/capping_migrate.py
2. 命令解释
   实时监控服务器功率，当某个机器的功率高于最大功率的一个百分比时，首先选择num个pod，执行迁移；迁移完成后仍无法降低功率则直接进行功率限制。
3. 返回结果格式

```
{ "pods": [ { "podName":"namexx1", "nameSpace":"default", "nodeName":"node14" }, {}, { "podName":"namexx2", "nameSpace":"default", "nodeName":"node15" } ]
```

1. 执行结果

```
{"pods": [{"podName": "pi-7jtg9", "nameSpace": "default", "nodeName": "k8s-master"}]}
```

```
当前节点的cap状态为： uncapping
当前节点的实时功率为： 80
k8s-master的状态为： uncapping
k8s-node的状态为： uncapping
以下节点的状态为uncapping，可以向上调度任务
k8s-master
k8s-node 节点上的pod pi-4fwh2需要被迁移走，应该迁移到k8s-master 上
第1次迁移正在进行。。。
第1次迁移完成
当前节点的cap状态为： uncapping
当前节点的实时功率为： 40
k8s-master的状态为： uncapping
k8s-node的状态为： uncapping
以下节点的状态为uncapping，可以向上调度任务
k8s-master
k8s-node 节点上的pod pi-rtc8n需要被迁移走，应该迁移到k8s-master 上
第2次迁移正在进行。。。
第2次迁移完成
当前节点的cap状态为： uncapping
当前节点的实时功率为： 20
功率足够小，停止迁移
```

# 五 scheduler extender配置及运行

## 配置

创建sched.yaml文件，保存scheduler extender的配置

```
apiVersion: kubescheduler.config.k8s.io/v1alpha2
kind: KubeSchedulerConfiguration
extenders:
  - urlPrefix: http://localhost:8888/
    filterVerb: filter
    prioritizeVerb: prioritize
    nodeCacheCapable: false
    enableHttps: false
    weight: 100000
leaderElection:
    leaderElect: true
clientConnection:
    kubeconfig: /etc/kubernetes/scheduler.conf
```

修改k8s调度器配置文件，一般在目录/etc/kubernetes/manifests下面

```
 containers:
 - command:
   - kube-scheduler
   - --authentication-kubeconfig=/etc/kubernetes/scheduler.conf
   - --authorization-kubeconfig=/etc/kubernetes/scheduler.conf
   - --bind-address=127.0.0.1
   - --kubeconfig=/etc/kubernetes/scheduler.conf
   - --leader-elect=true
   - --config=/etc/kubernetes/sched.yaml
   - --port=0
```

```
   volumeMounts:
   - mountPath: /etc/kubernetes/scheduler.conf
     name: kubeconfig
     readOnly: true
   - mountPath: /etc/kubernetes/sched.yaml
     name: schedconfig
     readOnly: true
```

```
  volumes:
  - hostPath:
      path: /etc/kubernetes/scheduler.conf
      type: FileOrCreate
    name: kubeconfig
  - hostPath:
      path: /etc/kubernetes/sched.yaml
      type: FileOrCreate
    name: schedconfig
```

## 运行

go build -o main *.go

./main