SCHEDULING CONTAINERS WITH KUBERNETES

Is it that different than other schedulers?
OUTLINE

- Kubernetes
- Other “Orchestrators”
- A flash of the past
- Scheduling in Kubernetes
BORG
KUBERNETES ARCHITECTURE

Diagram showing the architecture of Kubernetes, including components like kubectl, Devops, Master, api server, Scheduler, Replication controller, Node, kubectl, pod, Docker Registry, and Internet.
OPEN STACK ARCHITECTURE

Diagram showing the architecture of OpenStack, including components such as Network Node, Compute Node, and Cloud Controller Node, connected through various networks including Private Network, Data Network, and Public Network.
OPEN NEBULA
CONDOR HTC
ALL SIMILAR

- schedule virtual machines
- schedule containers
- schedule batch jobs

- serve requests based on available service center
DIFFERENCES

- Date of Birth
- Language
- API
- Workload or “intent”
SCHEDULING 101

- who can serve the request?
  - returns list of choices
  - predicates to filter “nodes”
- who has the highest priority?
  - returns ranked list, pick one
  - priority functions aka ranking
SCHEDULING PERFORMANCE

Pod startup latency

- 50th percentile
- 90th percentile
- 99th percentile

BACK TO CONDOR

- ClassAds for **matchmaking**.
- Before Kubernetes labels

("ClassAds are a flexible mechanism for representing the characteristics and constraints of machines and jobs in the Condor system"")

```plaintext
MyType = "Machine"
TargetType = "Job"
Machine = "froth.cs.wisc.edu"
Arch = "INTEL"
OpSys = "LINUX"
Disk = 35882
Memory = 128
KeyboardIdle = 173
LoadAvg = 0.1000
Requirements = TARGET.Owner="smith" || LoadAvg<=0.3 && KeyboardIdle>15*60
```
CONDOR “JOBS”

- job requirements and user defined ranking

\[
\text{Requirements} = \text{Arch}=="\text{INTEL}\" \&\& \text{OpSys} == "\text{LINUX}\"
\]
\[
\text{Rank} = \text{TARGET.Memory} + \text{TARGET.Mips}
\]

- Machine “affinity”

\[
\text{Friend} = \text{Owner} == "\text{tannenba}\" \mid\mid \text{Owner} == "\text{wright}\"
\]
\[
\text{ResearchGroup} = \text{Owner} == "\text{jbasney}\" \mid\mid \text{Owner} == "\text{raman}\"
\]
\[
\text{Trusted} = \text{Owner} != "\text{rival}\" \&\& \text{Owner} != "\text{riffraff}\"
\]
\[
\text{START} = \text{Trusted} \&\& ( \text{ResearchGroup} \mid\mid \text{LoadAvg} < 0.3 \&\& \text{KeyboardIdle} > 15*60 )
\]
\[
\text{RANK} = \text{Friend} + \text{ResearchGroup} \times 10
\]
KUBERNETES SCHEDULER

- make scheduler

- docker run --rm gcr.io/google_containers/hyperkube:v1.6.1 /hyperkube scheduler --help
SCHEDULING POLICY

- Default policy can be changed:

```json
{
    "kind": "Policy",
    "apiVersion": "v1",
    "predicates": [
        {
            "name": "PodFitsHostPorts",
        },
        {
            "name": "PodFitsResources",
        },
        ...
    ],
    "priorities": [
        {
            "name": "LeastRequestedPriority",
            "weight": 1,
        },
        ...
    ]
}
```

https://github.com/kubernetes/community/blob/master/contributors/devel/scheduler_algorithm.md
apiVersion: v1
class: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/e2e-az-name
                operator: In
                values:
                  - e2e-az1
                  - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
              - key: another-node-label-key
                operator: In
                values:
                  - another-node-label-value
containers:
  - name: with-node-affinity
    image: gcr.io/google_containers/pause:2.0
CUSTOM SCHEDULER

- Run multiple scheduler (different policies)
- Pods can specify which scheduler should schedule them

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  schedulerName: foobar
containers:
  - name: redis
    image: redis
```
```python
def main():
    w = watch.Watch()
    for event in w.stream(v1.list_namespaced_pod, "default"):
        if event['object'].status.phase == "Pending" and event['object'].spec.scheduler_name == scheduler_name:
            try:
                res = scheduler(event['object'].metadata.name, random.choice(nodes_available()))
            except client.rest.ApiException as e:
                print json.loads(e.body)["message"]
```

https://gist.github.com/sebgoa/c818bc46d33022c4bcd9d68aca0182f4
TIME FOR LITTLE DEMO?
EXPERIMENT

- take a 1000 node cluster
- Schedule 1 million jobs with Condor check Throughput
- Do the same with Kubernetes and the Job Object
- Compare :)
CONCLUSIONS

- Difference is API and abstractions
- Difference is the language
- Difference is Intent

- While you can use an existing system to run new types of application packages, you are retrofitting its initial intent and loosing out on the new abstractions
THANKS - @SEBGOA