CFN-dyncast
Load Balancing the Edges via the Network

Bing Liu, Jianwei Mao (speaker), Ling Xu, Ruizhao Hu, Xia Chen
2021.03.29
**Edge Computing and MEC sites**

**What is in a MEC site?**

**Central Cloud (Data center)**

**Bearer network**

**Edge Cloud (MEC site)**

**Access Network**

**Processing data at the edge**

- Better user experience
- Saving network bandwidth
- Limit the spread domain of sensitive data

**What is in a MEC site?**

- Limited physical size
- Not easy to scale-out
- Scale-up needs investment

**Routers and Switches**

**NFV servers**

**Application servers**

**Power supply**

**Cooling system**
Why Load Balancing is needed for the edges?

MEC sites as silos

I’m saturated!

We can’t help...

MEC sites as a pool

Thanks, guys!

MEC
Some Current Methods

DNS-based Load Balancer

- Not aware of the network status between the client and the site.

Layer 4 Load Balancer

- Not designed for servers across the WAN

Service status monitoring

- Go to site B
- www.example.com

Site A  Site B  Site C

I need

DNS

Layer 4 Load Balancer

- curl 100.100.100.100
- IPVS-Virtual Service 100.100.100.100
- Server 1 172.17.0.2:8000
- Server 2 172.17.0.3:8000
- Server 3 172.17.0.4:8000
What is CFN-dyncast trying to do?

How about using the network as a large distributed load balancer?

- Be aware of the load of the MEC sites and the network conditions
- Dispatch the client’s requests dynamically to the optimal MEC site
- Transparent to the client

Design considerations

- Compute status acquisition with application granularity
- Compute status advertisement
- Backward compatibility
- Session affinity
How does it work?

<table>
<thead>
<tr>
<th>IP Prefix</th>
<th>Next hop</th>
<th>Compute Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>APP1</td>
<td>MEC-1</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>MEC-2</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>MEC-3</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>MEC-n</td>
<td>35</td>
</tr>
</tbody>
</table>

1. Arrival of client’s demand for APP1
2. Ingress CFN router selects the MEC site to dispatch the demand, according to the compute metrics and network conditions
3. Establishing the session table entry
4. Update of LB table entry
5. Delivery to one of the application instances
6. Compute Status Acquisition and Compute Metric Calculation
7. Compute Metric Advertisement from MEC cloud platform to local CFN router
Compute Status Acquisition

- Kubelet
  - CPU usage
  - Memory usage
  - APP specific metrics
- Kube-state-metrics
  - CPU quota
  - Memory quota
- App Containers
- Prometheus
- Station Daemon
- BGP Speaker

Data aggregation per App
Calculation, Filtering, Trigger, etc.
Use BGP to advertise the compute metric, thus only the edge network devices need to be upgraded.
Session Affinity

**Session Table**

- Built at the ingress CFN router
- An entry is added when the dispatching decision is made
- The subsequent packets are sent to the MEC site as indicated by the table entry

<table>
<thead>
<tr>
<th>Session Identifier</th>
<th>Egress</th>
<th>Timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_IP</td>
<td>DST_IP</td>
<td>SRC_PORT</td>
</tr>
<tr>
<td>Client1</td>
<td>APP1</td>
<td>aaaa</td>
</tr>
<tr>
<td>Client2</td>
<td>APP2</td>
<td>cccc</td>
</tr>
</tbody>
</table>
The Experiment Environment
## Three LB methods in the Experiment

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
<th>Destination IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native DNS</td>
<td>The weights of the records are set statically based on the capacities of MEC sites. The weight is set to 2:1:1 on our testbed.</td>
<td>IP address of a certain MEC site</td>
</tr>
<tr>
<td>Compute-aware DNS</td>
<td>The weights of the records are set dynamically, based on the compute metric.</td>
<td>IP address of a certain MEC site</td>
</tr>
<tr>
<td>CFN-Dyncast</td>
<td>All the instances of an application are hidden behind an anycast IP, the MEC sites’ addresses are not visible by the clients. The CFN routers are responsible to dispatch the clients’ demands.</td>
<td>Anycast IP</td>
</tr>
</tbody>
</table>
Results

Job Completion Time (JCT)
The time duration that the client has to spend until it gets the response from the server.

<table>
<thead>
<tr>
<th></th>
<th>Native DNS</th>
<th>Compute-aware DNS</th>
<th>CFN-dynacast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average JCT (ms)</td>
<td>175.87</td>
<td>179.07</td>
<td>149.54</td>
</tr>
<tr>
<td>Span between the upper and lower JCT bound (ms)</td>
<td>110.259</td>
<td>95.449</td>
<td>72.256</td>
</tr>
<tr>
<td>No. of completed job within 20min</td>
<td>7236</td>
<td>7278</td>
<td>7989</td>
</tr>
</tbody>
</table>
Analysis

Why CFN-dyn­cast per­forms bet­ter than DNS-based schemes?

a) CFN just spends time on at most a single DNS request. And for compute-aware DNS, the client has to initiate DNS request once its local DNS cache expires, which add extra time to the JCT.

b) The optimal MEC site in a client’s local DNS cache can be outdated. Before the cache expires, the actual optimal MEC changes to another one, while the client keeps sending requests to an outdated and suboptimal site, which leads to longer JCT.
Conclusions and future works

- CFN-dyncast, a technique that aims to load balance the MEC sites in consideration of the compute status in application granularity and the network conditions.

- Evaluation result shows that
  - CFN-dyncast is capable to dynamically maintain the load of different MEC sites at the same level.
  - Compared to centralized dispatching schemes, CFN-dyncast helps the clients get replied by the servers in a shorten period.

- Next steps, we will try to evaluate CFN-dyncast on wide area network, in which the network condition could be a more remarkable element.
Thank you!