OVSDB Query Optimizer and Key-Value Interface

OVSCOn December 8-10, 2020 | Dmitry Yusupov
The need for scalability

Elasticity of SDN control plane?

- Kubernetes as of v1.19 supports 5000 nodes in production
- Large topologies with 1000+ EPs, LBs, Namespaces, Policies?
- What about scaling beyond 5000, can we get to 10,000 HVs?

- Can we elastically distribute SDN topology compute and
decentralized storage access as cluster growing?
OVN resourcing

Using different SKUs for central and HVs?

- Central components such as NorthD, RAFT OVSDB, CNI master can run on higher performance SKUs

- HV controllers can run on low profile SKUs, e.g. ARM devices with limited CPU and memory

- Can be beneficial for large DPU deployments, high-latency Edge IoT networks
A deeper look at OVSDB

Current OVSDB design thoughts

- Emphasis on read I/O scalability with dynamic distributed caches, side effect - stale reads

- Simplistic RAFT-based cluster for HA, side effect - no read after write guarantee, slow writes

- In-memory, unique relational database with only UUID-based query optimizer
Enhanced OVSDB Query Optimizer

Evolution of UUID-based optimizer

- Introduced Primary and Alternate key indexes [1]
- Reusing existing HMAP data structures
- Low overhead - 16 bytes per indexed key
- Results optionally can be ordered
OVSDDB Primary key design

Evolution of UUID-based optimizer

- There is no OVSDDB schema change
- Using existing per-table “indexes” keyword works well as it has to be unique
OVSDB Alternate key design

Evolution of UUID-based optimizer

- There is no such construct in OVSDB as of yet
- New boolean flag “alternate_key”: [true | false] introduced
- Alternate key implementation can use b-tree to enable ordered results, e.g. “ordered”:[“asc” | ”desc”]
OVSDB with Primary key performance
Measuring impact of using Primary key with small tables

Table size: 4,000 rows. In microseconds. No RAFT.

<table>
<thead>
<tr>
<th></th>
<th>Update with --may-exist</th>
<th>Find</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current code</td>
<td>265</td>
<td>277</td>
<td>119</td>
</tr>
<tr>
<td>With Query Optimizer</td>
<td>89</td>
<td>104</td>
<td>75</td>
</tr>
</tbody>
</table>

- Linear scan avoided
- $O(1)$ instead of $O(N)$
OVSDDB with Primary key performance
Measuring impact of using Primary key with larger tables

Table size: 60,000 rows. In microseconds. No RAFT.

<table>
<thead>
<tr>
<th></th>
<th>Update with --may-exist</th>
<th>Find</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current code</td>
<td>6700</td>
<td>5270</td>
<td>5200</td>
</tr>
<tr>
<td>With Query Optimizer</td>
<td>123</td>
<td>105</td>
<td>84</td>
</tr>
</tbody>
</table>

● Linear scan avoided
● $O(1)$ instead of $O(N)$
● Larger tables bigger impact
Benefits of enabling Query Optimizer in OVN

Primary Key
- Helps with Updates and Selects performance
- OVN Northbound database performance benefits the most
- OVN Southbound database performance improved when custom monitors are used

Alternate Key
- Helps with complex Select queries performance
- Only when user or application executes non-UUID based complex queries
Benefits of enabling Query Optimizer in OVN

If nothing else changed, just Primary key

- Linear scans are $O(N)$ expensive, can we optimize it out a bit? Yes!

- OVN Northbound database can benefit transparently when enabled as below (query_primary would be linear search coverage counter):

<table>
<thead>
<tr>
<th>query_linear</th>
<th>0.0/sec</th>
<th>0.017/sec</th>
<th>0.0028/sec</th>
<th>total: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_uuid</td>
<td>30.8/sec</td>
<td>23.333/sec</td>
<td>0.4628/sec</td>
<td>total: 1666</td>
</tr>
<tr>
<td>query_primary</td>
<td><strong>10.4/sec</strong></td>
<td><strong>5.717/sec</strong></td>
<td><strong>0.0972/sec</strong></td>
<td><strong>total: 350</strong></td>
</tr>
</tbody>
</table>
Applicability of Query Optimizer

An example of using it with Key-Value interface

- Benefits of KV interface are in simplicity and scalability
- OVSDB is a great piece of software, so, why not to try?
- OVSKV - a library that is layered on top of libovsdb [2]
- Compatible with ETCD like hierarchical key queries, e.g. /a/b/c* => value
Comparison of ETCD and OVSKV

Using fperf open source performance toolkit [4]

Table size: 60,000 keys. In microseconds.

<table>
<thead>
<tr>
<th></th>
<th>PUT</th>
<th>GET</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETCD</td>
<td>20700</td>
<td>230</td>
<td>14000</td>
</tr>
<tr>
<td>OVSD with KV</td>
<td>123</td>
<td>105</td>
<td>84</td>
</tr>
</tbody>
</table>

- 1-node OVSD
- Using fperf OVSKV backend [3]
Comparison of ETCD and OVSKV
Using fperf open source performance toolkit [4]

Table size: 60,000 keys. In microseconds.

<table>
<thead>
<tr>
<th></th>
<th>PUT</th>
<th>GET</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETCD</td>
<td>33400</td>
<td>415</td>
<td>18300</td>
</tr>
<tr>
<td>OVSD with KV</td>
<td>21400</td>
<td>105</td>
<td>14500</td>
</tr>
</tbody>
</table>

- 3-node OVSD

- Using fperf OVSKV backend [3]
Future work and ideas

What’s next?

- Ordered Alternate key work needs to introduce b-tree implementation
- In the perspective of recent DDlog work, can we scale out computation with D3log?
- Perhaps we can think of introducing multi-writer design to OVSDB?
- What if we switch to Key-Value interfaces? Maybe just for some tables?
Links and References

Show us the code...

1. Primary key implementation OVS github repo
   https://github.com/dyusupov/ovs/tree/query-optimizer-v1

2. OVSKV library github repo
   https://github.com/dyusupov/ovskv

3. fperf for OVSKV backend github repo
   https://github.com/dyusupov/fperf/tree/ovskv

4. fperf github repo
   https://github.com/fperf/fperf