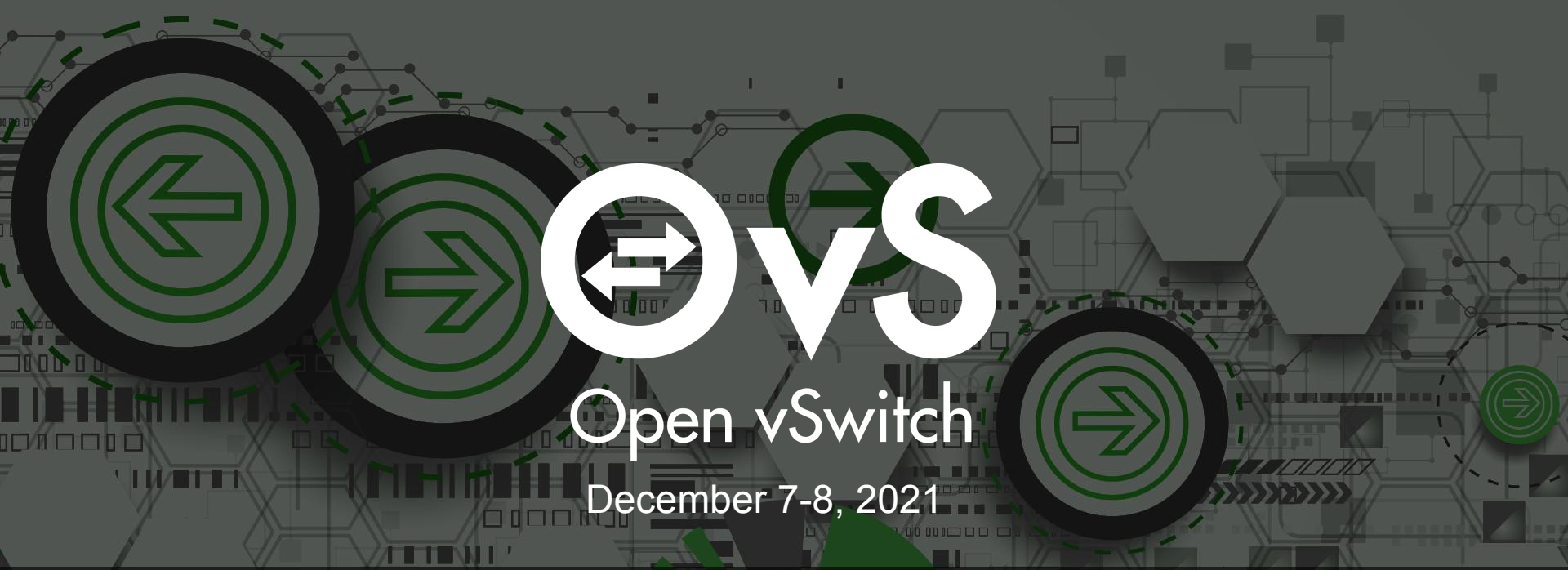
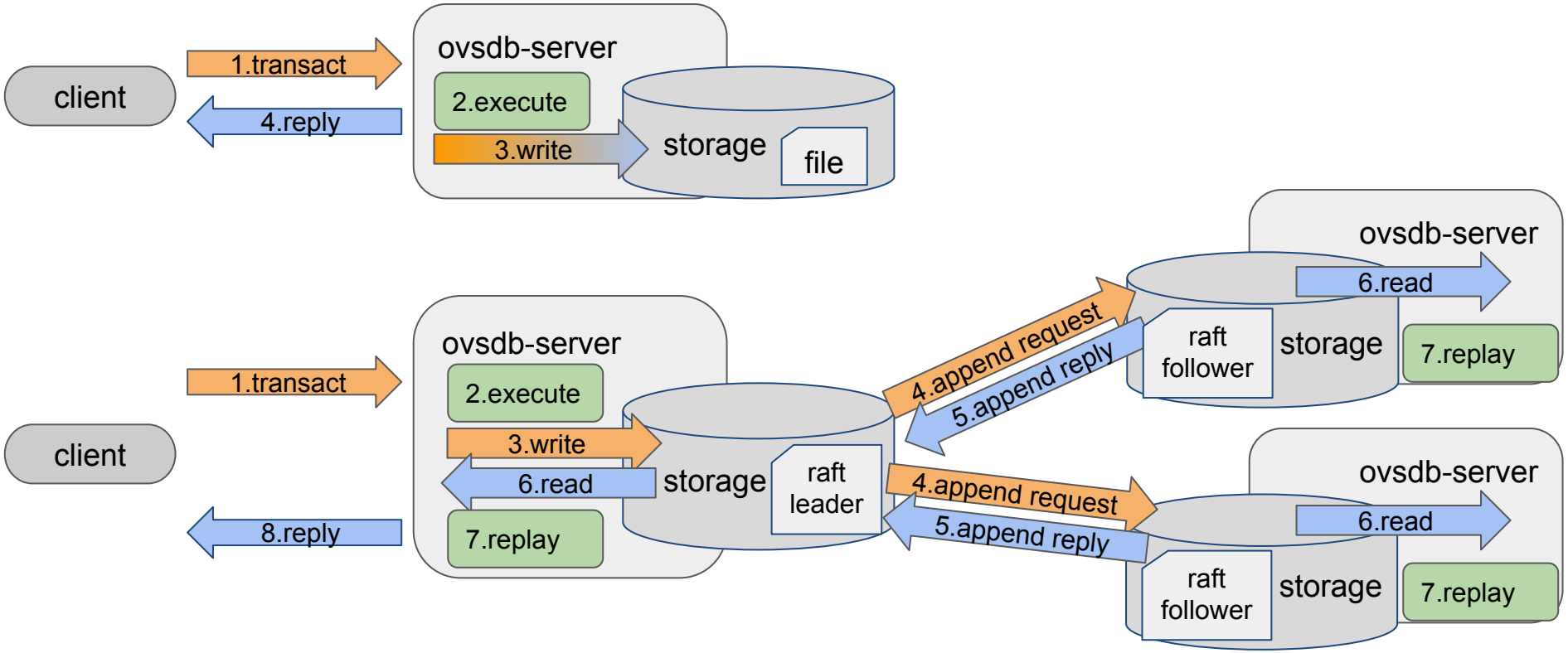


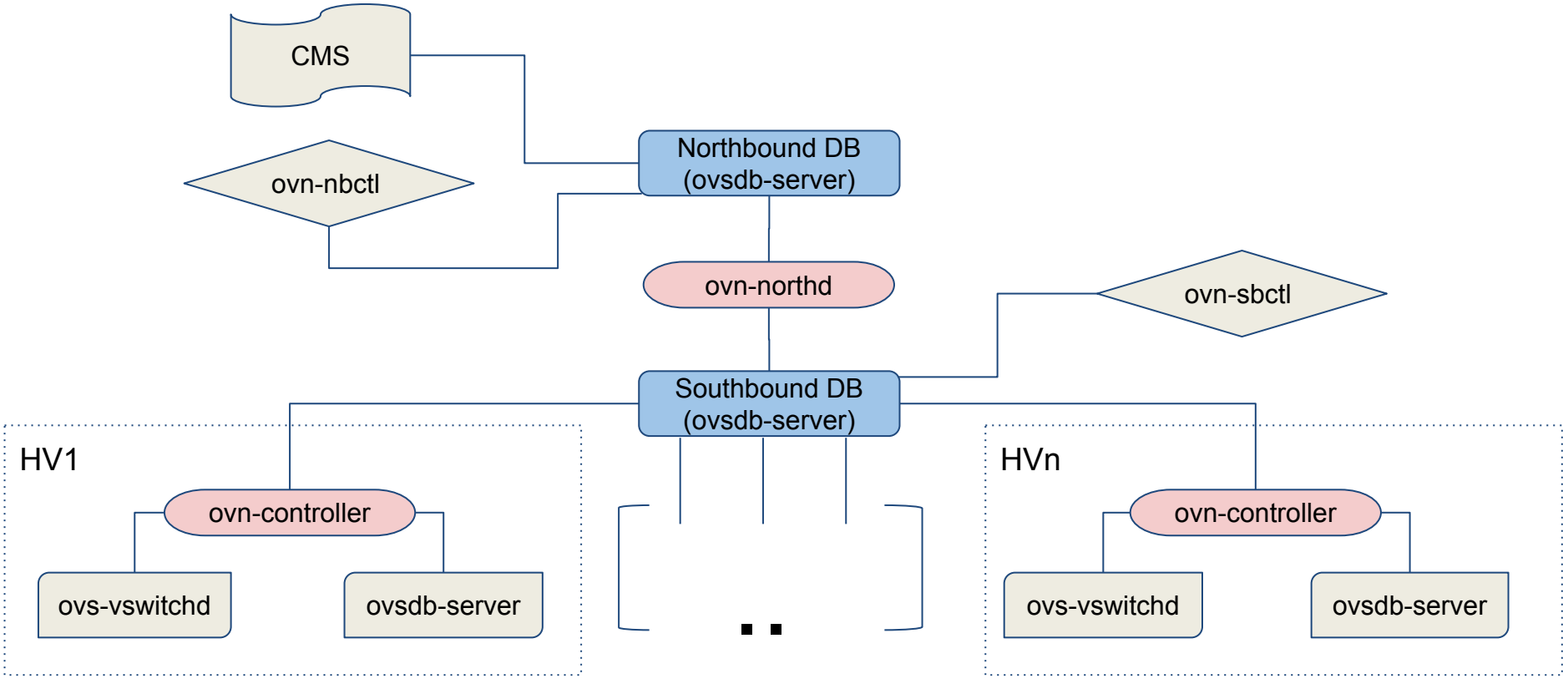
OVSDB: Performance and Scale Journey '21

Ilya Maximets, Red Hat

OVSDB: Standalone vs Clustered



OVN Architecture Overview



List of OVSDB performance changes

1. v2.13.5 - Baseline
2. 2ccd66f59 | ovsdb: Use column diffs for ovsdb and raft log entries.
3. 748010ff3 | json: Optimize string serialization.
4. 0de882954 | raft: Don't keep full json objects in memory if no longer needed.
5. 43e66fc27 | ovsdb: monitor: Store serialized json in a json cache.
6. b2712d026 | ovsdb: transaction: Use diffs for strong reference counting.
7. 51946d222 | ovsdb-data: Optimize union of sets.
8. bb12b6317 | ovsdb-data: Optimize subtraction of sets.
9. 32b51326e | ovsdb-data: Add function to apply diff in-place.
10. 429b114c5 | ovsdb-data: Deduplicate string atoms.
11. 4dbff9f0a | ovsdb: transaction: Incremental reassessment of weak refs.
12. 317b1bfd7 | ovsdb: Don't let transaction history grow larger than the database.
13. dec429168 | ovsdb-data: Consolidate ovsdb atom and json strings.
14. 79953a57e | stream-ssl: Avoid unnecessary memory copies on send.

Test setup and what the numbers mean

- ovsdb-server is running with OVN_Northbound schema in clustered or standalone mode.
- Test is sending transactions by invoking ovsdb-client application in leader-only mode.
- ovsdb-clients are invoked from 50 shells in parallel.
- Another 100 clients are monitoring the database receiving all the updates.
- Test is trying to perform operations that are similar to what ovn-kubernetes may do:
  - Port addition. Transaction with 3 operations (~750 B of data):
    - Create a row in a Logical_Switch_Port table
    - Add this row to one of the 100 Logical Switches
    - Add the address to the address set
  - Addition of a Load Balancer. Transaction with 2 operations (~420 B of data):
    - Create a row in a Load_Balancer table
    - Add this row to the Load Balancer Group
- 100 logical switches, 1 load balancer group and 1 address set are pre-created.
- Test is trying to consecutively add 30.000 LSPs (300 per LS) and 30.000 LBs, then remove.
  120.000 ovsdb-client invocations total.

Test setup and what the numbers mean

The metric for the test results is "**ovsdb-client invocations per second**".

Why **not** "transactions per second"?

- **Too many things** are happening beside the actual transaction:
  - ovsdb-client process needs time to **spawn/terminate/parse arguments**.
  - ovsdb-client needs to **connect** to the server and **request the Database record** from the _Server database (including 14KB OVN_Northbound schema).
  - By receiving the Database record, ovsdb-client may decide to **re-connect** to another server (leader-only).
  - ovsdb-client may decide to **backoff** in case of a connection failure (for the test backoff is limited by 0.5 seconds).

Baseline (v2.13.5) problem

- Standalone:
    - Rate: **40.9** invocations per second.
    - DB file size: **53 GB**
    - RSS: **87 MB**
- Clustered:
    - Rate: **DNF** (did not finish)
    - Projected DB file size at the end: **~70GB** per server.
    - RSS: **30 GB per process** at the moment it got killed (~ ⅓ of the test).

Problem: Fast growing change log in the database file and the RAFT log.

Transaction to add a port

How transactions look like in the database file:

```
"Logical_Switch":{"<r0>":{"ports":["uuid"," <p0>"]}}
"Logical_Switch":{"<r0>":{"ports":["set",[["uuid"," <p0>"],["uuid","<p1>"]]]}}
"Logical_Switch":{"<r0>":{"ports":["set",[["uuid"," <p0>"],["uuid","<p1>"],["uuid","<p2>"]]]}}
  ...
"Logical_Switch":{"<r0>":{"ports":["set",[["uuid"," <p0>"],...,["uuid","<p300>"]]]}}
```

Each file transaction **contains the whole new set** of values for the changed column.

Same for the RAFT log that is kept in memory → huge RSS of ovsdb-server process.

Baseline problem: Solution

OVS **2.15** supports **storing only the difference** between the old and new versions of a row:

```
#2 2ccd66f59 | ovsdb: Use column diffs for ovsdb and raft log entries.
```

Database file format changed in OVS 2.15, and it's **not backward compatible**.

For the **upgrade/downgrade** instructions see ovsdb(7):

https://docs.openvswitch.org/en/latest/ref/ovsdb.7

Test results v2.15.2 (column diff)

- Standalone:
    - Rate: **87.9** invocations per second. (was: **40.9**)
    - DB file size: **51 MB** (was: **53 GB**)
    - RSS: **79 MB**
- Clustered:
    - Rate: **13.7** invocations per second (was: v2.13.5 didn't finish the test)
    - DB file size: **~70 MB** per server (was: **~70 GB**).
    - RSS: **972 MB** (was: **> 30 GB**)

Conclusion:

Test was able to finish with a reasonable memory usage and **1000x** smaller database files.

Standalone database test finished with **2x** better performance result.

JSON-related optimizations

- Simple algorithmic change for the JSON serialization process that allows better compiler optimizations:

```
#3 748010ff3 | json: Optimize string serialization.
```

- Memory consumption optimization that allows to not store bulky JSON objects (lots of small memory pieces), but a single serialized string instead:

```
#4 0de882954 | raft: Don't keep full json objects in memory if no
                   longer needed.
```

- Pre-serialization of monitor replies to avoid doing the same thing for each ovsdb client:

```
#5 43e66fc27 | ovsdb: monitor: Store serialized json in a json cache.
```

JSON-related optimizations

- Since ovsdb-server and any other OVS-based application uses serialization of JSON objects for basically every interaction with other processes, this should have a noticeable performance impact.

- Unfortunately, current test framework doesn't add a lot of strings to the database.
  It's mostly UUIDs. So benefits from some patches are not very visible.

Alternative test: 100K

Same test as before, but instead of adding ports or load balancers, test sends **huge transactions, containing mostly strings**.

This is a better approximation of `OVN_Southbound` database, because:

- `ovn-northd` usually sends fairly big transactions
- `OVN Southbound` database contains a lot of strings, e.g. logical flow matches and actions.

What is actually going on:

- Each transaction in this test creates one port-group with 500 external ids.
- Each external id's key and value are 100B strings.
- Transaction size is about **100K bytes** long.
- Creating 5.000 port groups to not waste a disk space. (500 MB)

JSON-related: 100K test results

- Standalone:
  - #2: **35.7** invocations per second. RSS: **688 MB**
  - #5: **85.9** invocations per second. RSS: **688 MB**
- Clustered:
  - #2: **71.1** invocations per second. RSS: **1739 MB**
  - #5: **85.8** invocations per second. RSS: **1667 MB**

Conclusion:

1. 20% performance improvement in the clustered case and 140% improvement in the standalone case.
2. Performance improvement mostly caused by the patch #5.
3. RSS improvement is still not very visible, because database compactions are not happening during the test, hence we can't save on not storing a huge database snapshot in memory.

Strong reference counting

- In order to check referential integrity, each row has a reference counter.
- If value in a column changed, ovsdb need to decrease the counter for a row referenced by an old value and increase for the row referenced by a new value.
- What if the column contains a set of references?

Old column value:

a	b	c	d	e	f	g	h	i	j
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

New column value:

a	b	c	f	g	h	i	j	k	l
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1

Strong reference counting

- In order to check referential integrity, each row has a reference counter.
- If value in a column changed, ovsdb need to decrease the counter for a row referenced by an old value and increase for the row referenced by a new value.
- What if the column contains a set of references?

Old column value:

a	b	c	d	e	f	g	h	i	j
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

New column value:

a	b	c	f	g	h	i	j	k	l
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1

- What if that set has **30.000** values?

Strong reference counting: Solution

- Get the difference between sets and update only changed values!

Old column value:

a	b	c	d	e	f	g	h	i	j
			-1	-1					

New column value:

a	b	c	f	g	h	i	j	k	l
								+1	+1

- Fortunately, while processing transactions we either already know the difference or able to calculate it fairly fast.

```
#6 b2712d026 | ovsdb: transaction: Use diffs for strong reference counting.
```

Test results (strong refs, ports+LBs)

- Standalone:
  - #5: **90.7** invocations per second. RSS: **79 MB**
  - #6: **95.7** invocations per second. RSS: **78 MB**
- Clustered:
  - #5: **11.7** invocations per second. RSS: **916 MB**
  - #6: **9.8** invocations per second. RSS: **803 MB**

Analysis:

1. 5% performance improvement in the standalone case
2. 16% performance drop in the clustered case.???

Test results (strong refs, ports+LBs)

- Standalone:
  - #5: **90.7** invocations per second. RSS: **79 MB**
  - #6: **95.7** invocations per second. RSS: **78 MB**
- Clustered:
  - #5: **11.7** invocations per second. RSS: **916 MB**
  - #6: **9.8** invocations per second. RSS: **803 MB**

Analysis:

1. 5% performance improvement in the standalone case.
2. 16% performance drop in the clustered case.???

   Reason: patches wasn't applied in the order they were developed. At this point in the git history calculation of the column difference is a bit heavy and clustered database uses this path twice, hence gets the performance hit. Next patches will make these operations way cheaper, so #6 will shine in their performance results.

Optimized operations on sets

- Algorithmic changes to make set operations almost incremental:

```
#7 51946d222 | ovsdb-data: Optimize union of sets.
#8 bb12b6317 | ovsdb-data: Optimize subtraction of sets.
#9 32b51326e | ovsdb-data: Add function to apply diff in-place.
```

- `#7` and `#8` allows to get rid of extra qsort() of the set and separate copies of all the elements.

- `#9` is very similar and also allows to avoid extra memory copy while applying the column diff read from the database file/raft log by applying the diff to the existing row without constructing a new one.

- Detailed description of the algorithmic changes could be found in the commit messages.

Test results (set operations)

- Standalone:
  - #6: **95.7** invocations per second. RSS: **78 MB**
  - #9: **127.3** invocations per second. RSS: **79 MB**
- Clustered:
  - #6: **9.8** invocations per second. RSS: **803 MB**
  - #9: **52.1** invocations per second. RSS: **403 MB**

Analysis:

1. **33%** performance improvement in the standalone case.
2. **431%** performance and **50%** memory consumption improvement in the clustered case.

Deduplication of strings

- OVSDB copies database rows and columns all the time during transaction processing.
- This includes copying strings stored in the database back and forth.
- Let's use a reference-counted data structure instead:

```
#10 429b114c5 | ovsdb-data: Deduplicate string atoms.
```

- This change also allows to compare strings very fast as most of the time OVSDB actually compares strings that it previously copied, so the simple pointer comparison will confirm that they are identical.

Test results (string deduplication)

- Standalone:
  - #9: **127.3** invocations per second. RSS: **79 MB**
  - #10: **157.6** invocations per second. RSS: **93 MB**
- Clustered:
  - #9: **52.1** invocations per second. RSS: **403 MB**
  - #10: **82.0** invocations per second. RSS: **270 MB**

Analysis:

1. **24%** performance improvement in the standalone case.
2. **57%** performance and **33%** memory consumption improvement in the clustered case.
3. Memory consumption is slightly higher in a standalone case because actual data structure with a reference counter is a bit larger than just string and standalone db doesn't copy data that much.

Reassessment of weak references

- Since referenced row can be deleted, OVSDB has to store back-references in order to find the source row and clean up actual references.
- Previously, the row was marked for re-assessment once one of the references in it got removed (marked through the back-reference) or added (marked directly).
- Reassessment process **re-checked all the references** in a row.
- Following patch made it possible to only re-check references that was added or removed making the whole process more or less **incremental**:

```
#11 4dbff9f0a | ovsdb: transaction: Incremental reassessment of weak refs.
```

Test results (incremental weak refs)

- Standalone:
  - #10: **157.6** invocations per second. RSS: **93 MB**
  - #11: **274.5** invocations per second. RSS: **97 MB**
- Clustered:
  - #10: **82.0** invocations per second. RSS: **270 MB**
  - #11: **197.3** invocations per second. RSS: **266 MB**

Analysis:

1. **74%** performance improvement in the standalone case.
2. **140%** performance improvement in the clustered case.
3. LBs are using weak references, so performance in that part improved dramatically, since OVSDB doesn't need to re-check the whole Load Balancer Group on every transaction.

Transaction history limit

- Transaction history is a storage for recent transactions for the purpose of fast resync of clients on re-connection with monitor_cond_since (monitor v3).
- History can store up to a 100 transactions.
- Fast re-sync is not that fast if transaction history is actually larger than the database itself, so the size limit was introduced:

```
#12 317b1bfd7 | ovsdb: Don't let transaction history grow larger than the database.
```

Test results (history limit)

- Standalone:
  - #11: **274.5** invocations per second. RSS: **97 MB**
  - #12:**273.5** invocations per second. RSS: **96 MB**
- Clustered:
  - #11: **197.3** invocations per second. RSS: **266 MB**
  - #12:**185.0** invocations per second. RSS: **154 MB**

Analysis:

1. Performance in the clustered case degraded by 6% since the OVSDB frees memory more frequently.
2. At the same time the memory consumption in a clustered case reduced significantly. **42%** in this test scenario. And this can be much more significant in a real world case. In ovn-kubernetes scenarios without LB Groups this change showed up to 99% reduction in memory usage of OVN Northbound database.

Further consolidation of strings

- Continuation of string deduplication work started in patch #10

- This time we're re-using same strings for ovsdb objects and JSON objects in order to avoid unnecessary copies every time we converting them into each other:

```
#13 dec429168 | ovsdb-data: Consolidate ovsdb atom and json strings.
```

SSL and memory copy

- Problem:

  stream-ssl library performs a copy of the data before sending it every time.

- This last bit is to avoid the copy if not necessary:

```
#14 79953a57e | stream-ssl: Avoid unnecessary memory copies on send.
```

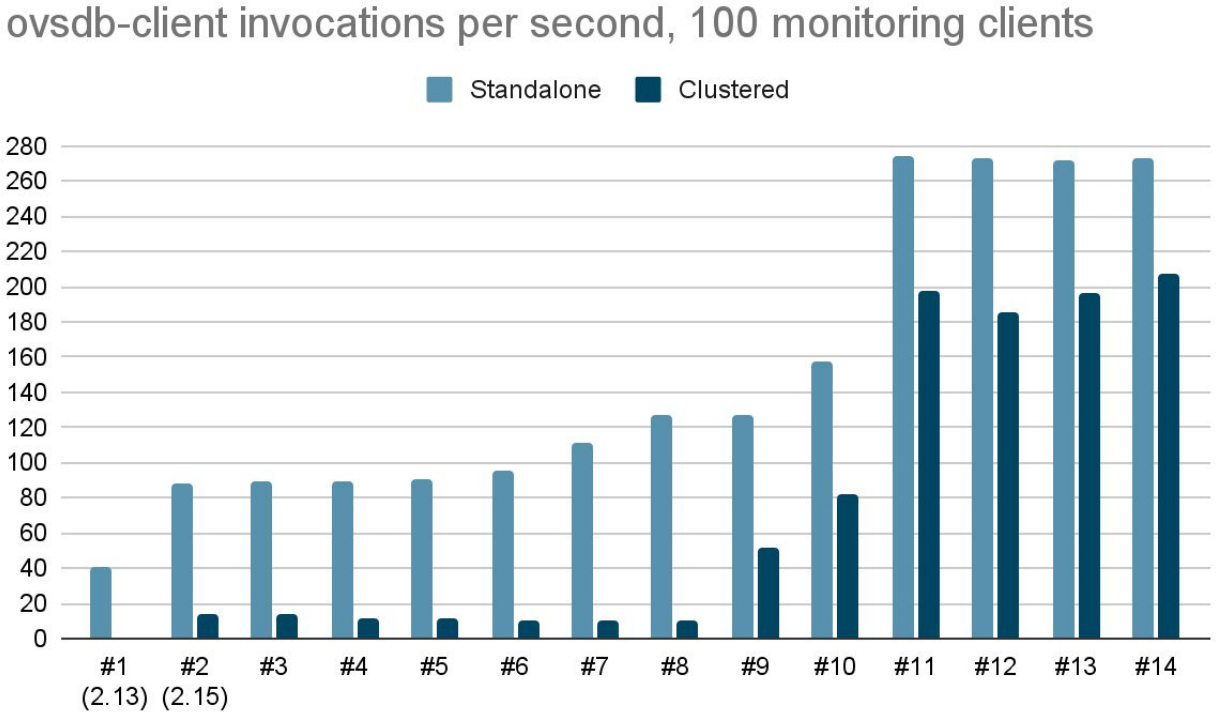
Test results (strings + ssl)

- Standalone:
  - #12: **273.5** invocations per second. RSS: **96 MB**
  - #14: **273.2** invocations per second. RSS: **102 MB**
- Clustered:
  - #12: **185.0** invocations per second. RSS: **154 MB**
  - #14: **207.0** invocations per second. RSS: **157 MB**

Analysis:

1. Again a slight memory consumption increase since the JSON structure is a bit larger than the one we used for the OVSDB atom.
2. **12%** performance improvement for the clustered database.

Test results



Standalone:

x3

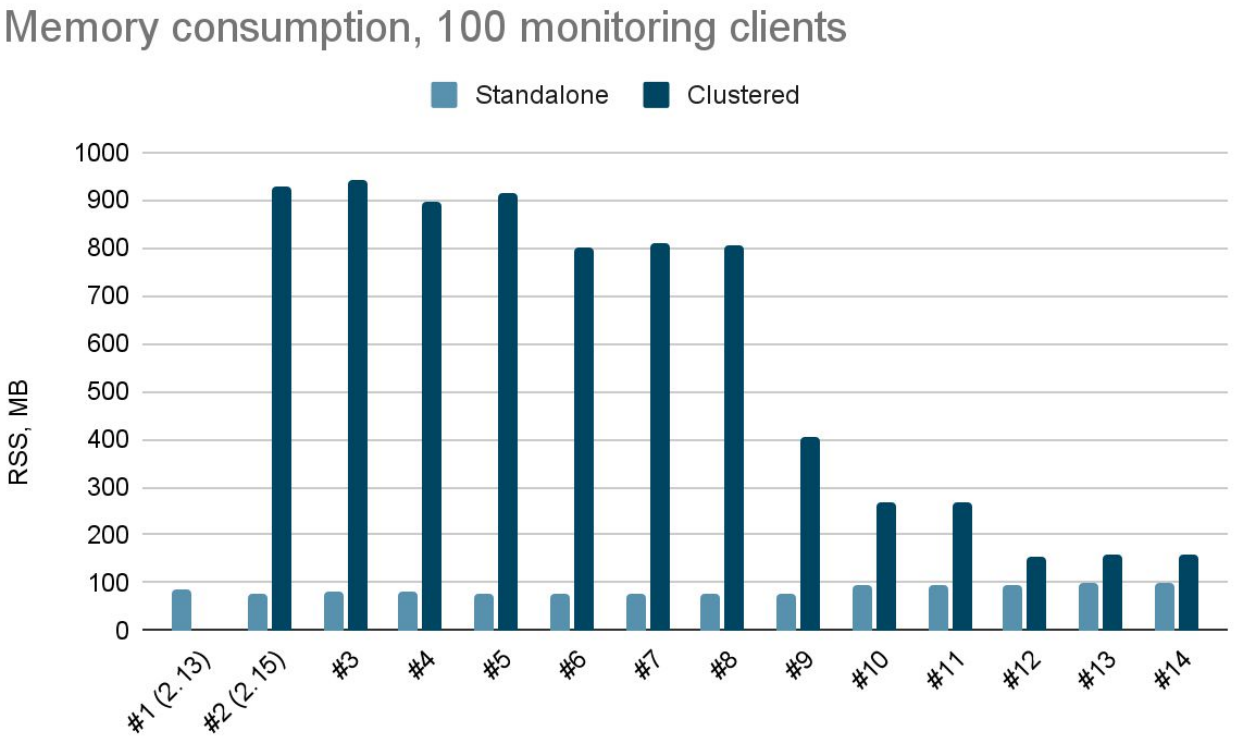
87.9 (v2.15) ⟹ 273.3

Clustered:

x15

13.7 (v2.15) ⟹ 207.0

Test results: RSS



Standalone:

87MB (v2.15) ⟹ 102MB

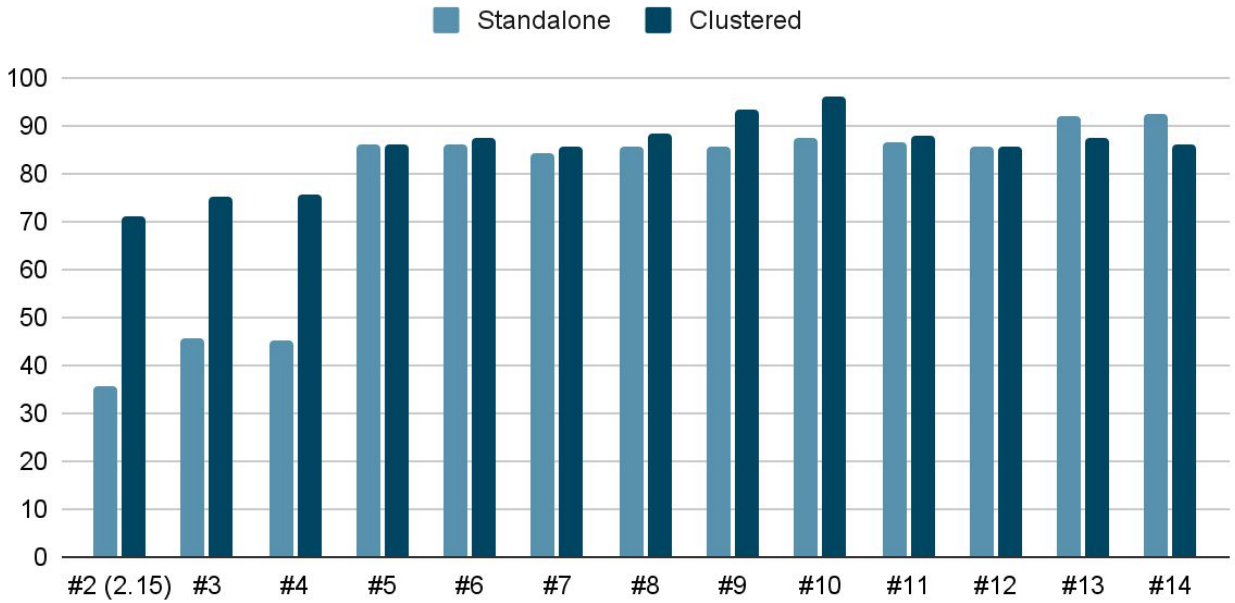
Clustered:

x6

927MB (v2.15) ⟹ 157MB

Test results: 100K test

ovsdb-client invocations per second, 100 monitoring clients, 100K test



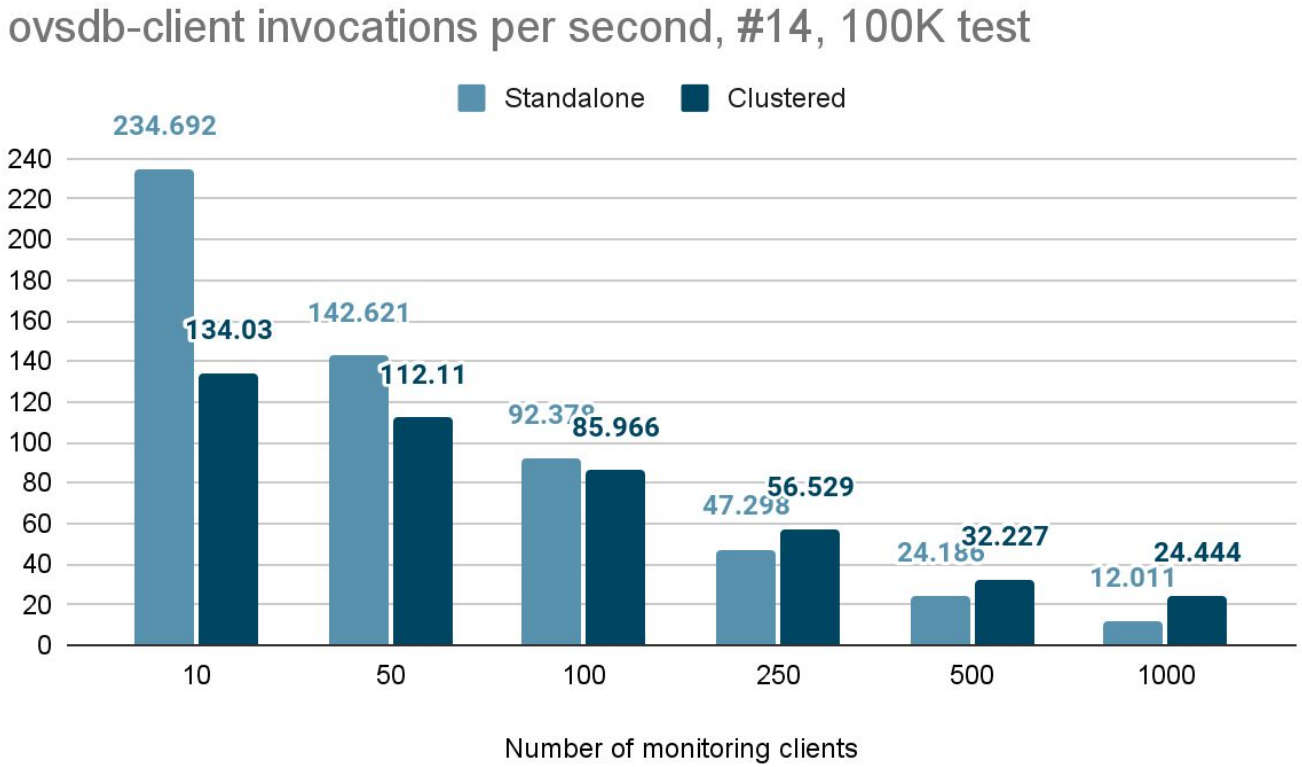
Standalone:

35.7 (v2.15) **x2.5** ⟹ 92.3

Clustered:

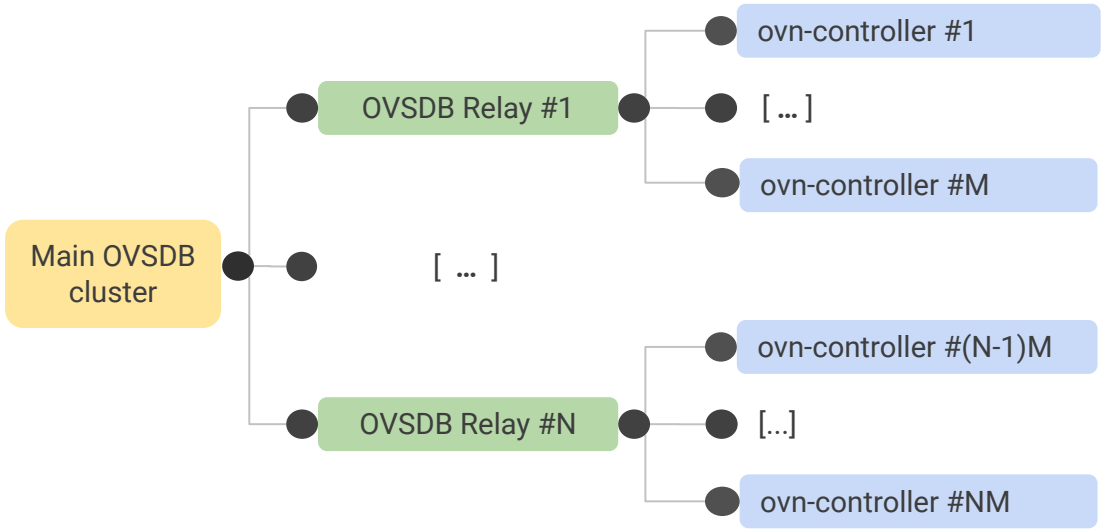
71.1 (v2.15) **x1.2** ⟹ 85.9

Test results: Number of monitors



N clients	Volume of monitor updates per second	
	Standalone	Clustered
10	223 MB	127 MB
50	680 MB	534 MB
100	880 MB	819 MB
250	1.1 GB	1.3 GB
500	1.1 GB	1.5 GB
1000	1.1 GB	2.3 GB

OVSDB Relay service model



- Relay is a client for a main database, but a server for actual clients.
- Relays can take a heavy lifting of delivering monitor updates.
- Not read-only!
- Introduced in OVS 2.16

To start a relay server that will accept connections on `<ip>:<port>` and will relay the `OVN_Southbound` database located at `<main-ip>:<main-port>`:

```
$ ovsdb-server --remote=ptcp:<port>:<ip> relay:OVN_Southbound:tcp:<main-ip>:<main-port>
```

More information: https://docs.openvswitch.org/en/latest/topics/ovsdb-relay/

Other important changes

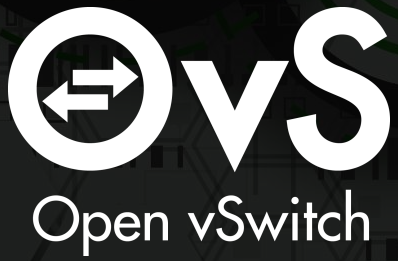
- OVSDB C IDL:
  - Performance improvements and bug fixes (Dumitru Ceara)
  - Implementation split into 2 separate modules (Ben Pfaff)
  - New APIs to check table/column existence (Numan Siddique)

- Python IDL bug fixes and performance optimizations (Terry Wilson)

- RAFT cluster stability enhancements.

Future plans

- Continue with incremental performance optimizations. There are still low hanging fruits to optimize.
- Database compaction on the background.
- Explore possible optimizations for conditional monitoring.
- Re-work the test framework used for this presentation so everyone can use it. Make it closer to “transactions per second” metric.
- Better configuration capabilities for relays (inactivity probes, etc.)



Thanks!

Email: i.maximets@ovn.org