Network and On-Disk Compression in YugabyteDB

Raghavendra T. K.

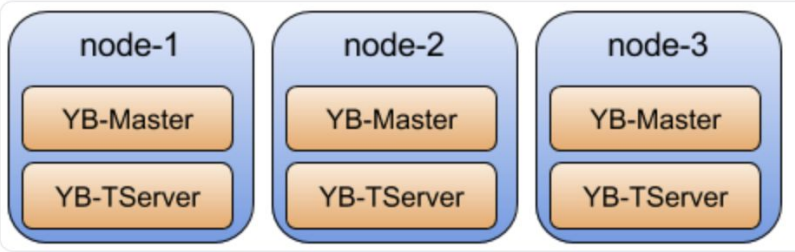
Friday, Jan/13/2023

YFTT YugabyteDB Friday Tech Talks

yugabyteDB

Network compression in YugabyteDB

Motivation: Cloud providers charge for data transfer across availability zones/regions. These costs can be significant.



Yugabyte has tons of traffic across nodes - communication between master and t-server, the heartbeat messages across nodes, **raft replication of the data**, reads that span several nodes. They all can benefit from network compression.

How to enable network/stream compression? Controlled by two tserver gflags:

```
--enable_stream_compression (Enables or disables stream compression)
Values : true (default), false
--stream_compression_algo (Specifies which RPC compression algorithm to use).
Values :  0 (No compression - default value), 1 (Gzip), 2 (Snappy), 3 (LZ4)
```

Network compression in YugabyteDB

Which compression strategy is better? Let's take a look at Sysbench numbers:

	Selects 24 connections | 250 tbls | 0.5M										C5L = $1,080 + Usage		
	Latency	Throughput	Network Packets RX(K)	Network Packets TX(K)	Network bandwidth RX(MB)	Network bandwidth TX(MB)	CPU (MAX)	Latency impact	Throughput impact	Compression rate	Gb / Mo	Network Cost Usage	Compression Savings
No compression	33.64	713.31	19.3	19.3	6	6.7	76	0	0	0.00	98,755	$1,975.10	
gzip	41.23	582.03	20	20	3.1	3.7	85	1.23	0.82	46.46	52,877	$1,057.54	$917.57
snappy	37.36	642.37	22	22	5.6	6.4	75	1.11	0.90	5.51	93,312	$1,866.24	$108.86
lz4	37.63	637.71	21	21	5.35	6	82	1.12	0.89	10.63	88,258	$1,765.15	$209.95

	Inserts 12 connections | 250 tbls | 0.5M										C5L = $1,080 + Usage		
	Latency	Throughput	Network Packets RX(K)	Network Packets TX(K)	Network bandwidth RX(MB)	Network bandwidth TX(MB)	CPU (MAX)	Latency impact	Throughput impact	Compression rate	Gb / Mo	Network Cost Usage	Compression Savings
No compression	90.95	131.93	18	18	5.8	5.6	82	0	0	0.00	88,646	$1,772.93	
gzip	143.91	83.38	19.7	19.7	3.37	3.16	89	1.58	0.63	48.58	50,777	$1,015.55	$757.38
snappy	137.00	87.59	20.3	20.3	5.9	5.7	86	1.51	0.66	8.66	90,202	$1,804.03	-$31.10
lz4	129.42	92.71	21	21	6.3	6.08	82	1.42	0.70	2.52	96,267	$1,925.34	-$152.41

Gzip shows much greater compression ratio than lz4, at the cost of higher CPU.

Network compression in YugabyteDB

Results from TPCC runs:

M6i 2XL instances | TPCC 500WH 300 connections | With and Without packed rows | YBM v2.15.1										
Cluster Type	Compression Type	TPCC output			YB op latencies | YB managed page			YB resources | YB managed page		
		Efficiency	Latency(ms)	Throughput	Selects(ms)	Inserts(ms)	Updates(ms)	CPU	Transmitted(Mb)	Received(Mb)
Without packed rows	No compressio	99.97	58.25	237.4	1.3	2.5	2.6	36	5.9	5.8
	gzip	99.95	75.07	236.88	1.7	3	3	44	2.9	2.9
	LZ4	99.86	64.81	236.64	1.5	2.7	2.8	39	5.2	5.3
With packed rows	No compressio	99.93	66.44	237.54	1.2	2.5	2.5	35	5.8	5.7
	gzip	99.86	70.39	236.93	1.6	2.86	3	44	3	2.92
	LZ4	99.91	64.04	237.32	1.23	2.44	2.56	36	5.3	5.1

- GZIP provides higher compression (40-50%) for both sysbench and TPCC workloads.
- GZIP uses up more CPU usage and thus lowers throughput.
- LZ4 provides 10-15% compression (10+% increased latencies) when compared to no-compression.

Network compression in YugabyteDB

Summary

The network compression strategy depends on the deployment **topology** and workload pattern.

- In Multi-AZ/region deployments, where network costs are high, use **gzip** as the compression library (slightly less throughput).
- In Single-AZ deployments, where the network cost is not incurred, use **lz4** or **disable** stream compression.

Yugabyte Managed (YBM) uses the above strategy.

Is it safe to enable? Available in all release and preview builds, since 2.6.3 release.

- Versions prior to 2.6.3 does not support it, so it should be enabled only after all the nodes in the cluster have successfully been upgraded to version 2.6.3 or above.

Storage compression in YugabyteDB

Motivation: Workloads that store 100s of GB or TB per node benefit a lot from storage compression.

DocDB layer in YugabyteDB stores data in SST files using RocksDB, supports three types of compression schemes.

How to use it? Controlled by tserver gflag:

```
--compression_type

Values: snappy (default), zlib, lz4, NoCompression
```

- If you select an invalid option, the cluster will not come up.
- Changing this flag on an existing database is supported.
- Changing the compression type takes effect after the nodes are restarted.
- A tablet can have SSTs with different compression types.
- If the compression type is changed, compaction will eventually remove the files compressed with older scheme.

Storage compression in YugabyteDB

Results from TPCC experiment (250 warehouses on a RF-3 cluster).

	Compression_type		
Results	**Snappy**	**Zlib**	**NoCompression**
Size of SST files	29.2 GB per node	20.2 GB per node	60 GB per node
TPM-C	3158.93	3160.27	3156.18
Efficiency	98.26%	98.30%	98.73%
Throughput (req/s)	114.98	115.04	114.9

Results from TPCC experiment (2500 warehouses on a RF-3 cluster).

	Compression_type		
Results	**Snappy**	**Zlib**	**NoCompression**
Size of SST files	293 GB per node	204 GB per node	606 GB per node
TPM-C	31943.83	31603.47	31691.8
Efficiency	99.36 %	98.30 %	99.73%
Throughput (req/s)	1172.58	1148.43	1134.9

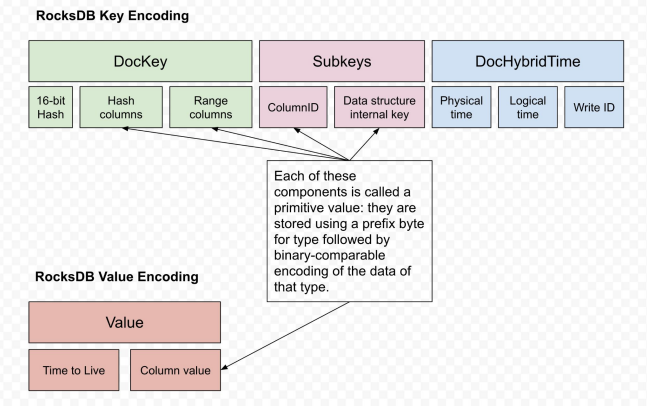
Storage compression in YugabyteDB

Observations

- Zlib compresses close to ⅓ the uncompressed size.
- Zlib provided 20+% space gains, when compared to Snappy across experiments.
- Overall CPU utilization with Zlib compression is higher (~3-5%) compared to Snappy across the board - TPCC runs, CassandraKeyValue (read intensive workloads).
- Compaction times in Zlib are 6-8x higher than Snappy compression (both had the same number of compactions during the period).

Delta Encoding in YugabyteDB

- Delta Encoding in RocksDb uses simple prefix encoding.
- This scheme doesn't work very well for Yugabyte because of the storage format.



- For SST files that are sorted, the repetition of key components consumes quite a bit of space. They benefit from delta encoding scheme.
- Yugabyte implemented three shared parts encoding scheme to take advantage of sub-parts of the key that can be common.

Delta Encoding in YugabyteDB

How to use it? Controlled by tserver gflag:

```
--regular_tablets_data_block_key_value_encoding
values - shared_prefix (default), three_shared_parts
```

Available from 2.8.0 release onwards.

- However, three_shared_parts encoding is default for new clusters starting from 2.16.0.
- Shared_prefix is a the default for clusters that are at 2.14 or below, the default encoding remains as and needs to be updated to three_shared_parts to get the benefit.

Delta Encoding in YugabyteDB

Results: SST file sizes of stock table (TPCC workload).

	Size of the SST file	
Compression	**Shared prefix encoding**	**Three parts encoding**
No compression	1.12 GB	775.60 MB
snappy	595.18 MB	508.39 MB
zlib	401.27 MB	347.61 MB

Summary

- The data in the block cache is in the encoded form, so the scheme helps reduce the memory footprint of the block cache too.
- Delta encoding + storage compression provides space gains for workloads.

References

- Advanced Delta Encoding design on Github

Thank You

Join us on Slack: yugabyte.com/slack (#yftt channel)

Star us on Github: github.com/yugabyte/yugabyte-db

YFTT YugabyteDB Friday Tech Talks

yugabyteDB