Packed Row Storage Format

Raghavendra TK

Friday, Aug/19/2022

YFTT YugabyteDB Friday Tech Talks

yugabyteDB

Current storage format in YugabyteDB

A row corresponding to the user table is stored as multiple key value pairs in the storage engine - DocDB. For example,

Table with structure:

K (PK)	C1	C2	.....	Cn
k1	10	20		1000

DocDB format:

< k1.C1 → 10 >
…
…
< k1.Cn → 1000 >

Motivation: Efficient updates: Only the columns being updated are locked, read and written. High degree of concurrency for updates, reduced write amplification for updates.

Side effects: Storage cost is high (even with prefix compression scheme). Not very efficient for bulk ingestion.

User defined types in YugabyteDB

Customers can pack the different columns using user defined types (UDT), which get stored as a single key value pair in DocDB.

< k1 → UDT { C1 → 10, C2 → 20, …… , Cn → 1000 } >

Side effects of UDTs

- DocDB (Storage layer) is not aware of the user columns, treats it like blob of bytes.
- UDTs are not UPDATE friendly - UPDATE of an attribute requires read-modify-write of entire UDT, resulting in sub-par performance.
- The updates cannot be pushed to the DocDB layer, and hence the performance of updates are impacted.

Why do we need Packed Row Storage format?

Overtime, the need for Packed Row has increased; Customers moving from traditional SQL environments are used to the benefits of such packed format:

- Lower storage footprint.
- Efficient INSERTs, especially when a table has large number of columns.
- Faster bulk ingestion.
- UDTs require application rewrite, are not necessarily an option for everyone, like latency sensitive update workloads.

What does Packed Row Storage look like ?

A row corresponding to the user table is stored as a single key-value pair in the storage engine - DocDB.

< k1 → Packed { C1 → 10, C2 → 20, …… , Cn → 1000 } >

DocDB is aware of the schema, can leverage flexible packing strategies.

Design aspects of Packed Row Storage

Inserts: Row is stored as a single key-value pair.

DocDB entry: < k1 → Packed { C1 → 10, C2 → 20, …… , Cn → 1000 } >

Updates: If some column(s) are updated, then each such column update is stored as a key-value pair in DocDb (same as without packed columns).

DocDB entries:
< k1 → Packed { C1 → 10, C2 → 20, …… , Cn → 1000 } >
< k1.Cn → 2000 >

If all non-key columns are updated, then the row is stored in the packed format as one single key-value pair.

DocDB entries:
< k1 → Packed { C1 → 10, C2 → 20, …… , Cn → 1000 } >
< k1 → Packed { C1 → 11, C2 → 21, …… , Cn → 1001 } >

This scheme adopts the best of both worlds - efficient updates and efficient storage.

Design aspects of Packed Row Storage (Continued)

Select: Scans need to construct the row from packed inserts as well as non-packed update(s) if any.

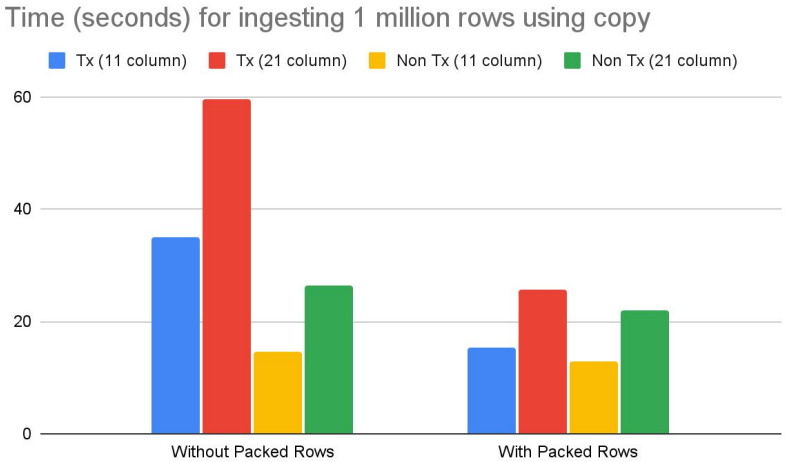
Compactions: Compactions produce a compact version of the row, if the row has unpacked fragments due to updates.

Before Compaction:	< k1 → Packed { C1 → 10, C2 → 20, …… , Cn → 1000 } > < k1.Cn →2000 >
After Compaction:	< k1 -> Packed { C1 → 10, C2 → 20, …… , Cn → 2000 } >

Backwards compatible! Read code can interpret non-packed format as well. Write/Updates can produce non-packed format as well.

Results: Throughput

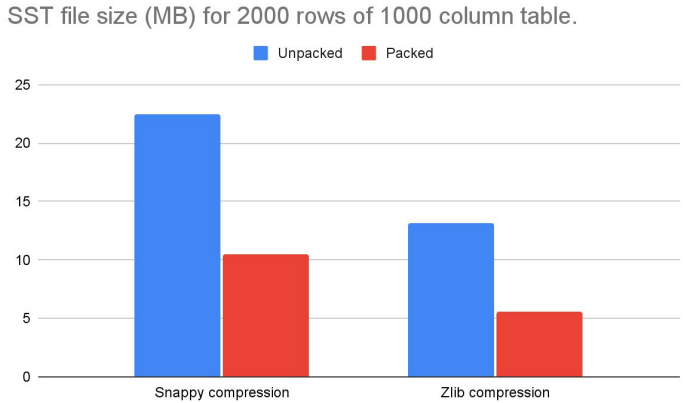
Experiment: Bulk load of 1 million rows using copy.



Summary: Bulk load is at least 2x faster in Packed versus non-packed. For tables with larger number of columns, the speed up increases is even higher (4-5x as reported by some customers). Packed Rows reduces the gap between transactional and non-transactional bulk loads.

Results: SST file sizes

Experiment: Insert 2000 rows in a 1000 column table.



Summary: On-disk sizes are about 2x better for Packed vs. Unpacked.

In memory (block cache) usage is 5x better (93MB vs.18MB).

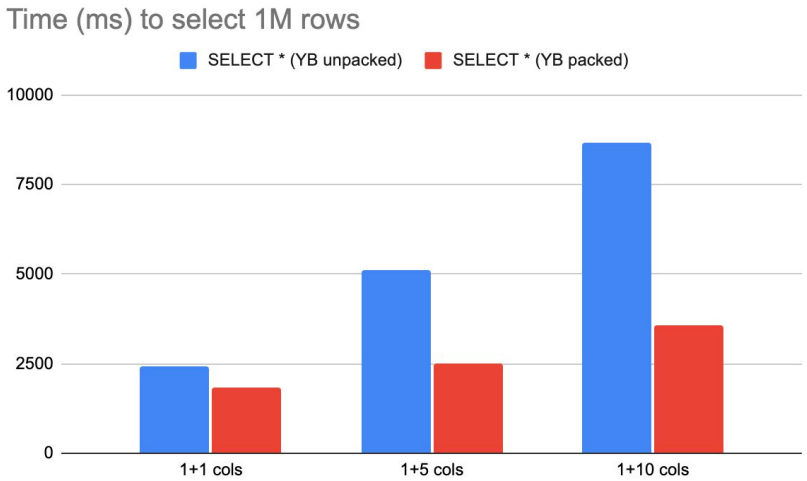
Demo

YFTT YugabyteDB Friday Tech Talks

yugabyteDB

Results: Scans

Scan performance: For sequential scan of table with 1 million rows, Packed columns is 2x better than non-packed case (even better for wider tables).



All in all, win-win for most workloads!

How to use Packed Row feature?

Packed Rows is available as a **Beta** feature in Release - **2.15.1.0-b175** onwards**.**

Enable tserver gflag **- ysql_enable_packed_row** on the universe.

Additional knobs/gflags:

ysql_packed_row_size_limit - Packed row size limit for YSQL, defaults to block size limit. For rows that are over the block size limit, such rows will be stored in unpacked form (like before).

Limitations / Roadmap

Now: 2.15.1 release - Packed rows feature in Beta.

- **Backwards compatible**: Feature ON produced packed format (for new inserts), Feature OFF produces unpacked format (new inserts).
- **Cross feature compatible**: Works with PITR, Co-located tables.
- Integration with xCluster and CDC (In progress) - There are some known limitations with xCluster and schema changes/DDLs and Packed Row feature.

Future: 2.17.X release

- Packed rows feature works well across features like xCluster, CDC.
- Packed rows support for YCQL.
- Additional storage optimizations - storing NULLs efficiently etc.

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