Columnstore and B+ tree – Are Hybrid Physical Designs Important?

Adam Dziedzic, Jingjing Wang, Sudipto Das, Bolin Ding, Vivek R. Narasayya, Manoj Syamala

Microsoft  The University of Chicago  University of Washington
Physical designs for diverse workloads

OLTP

B+ tree
Physical designs for diverse workloads

OLTP

Analytics

B+ tree

COL
Physical designs for diverse workloads

OLTP

B+ tree

Analytics

COL

B+ tree & Columnstore on same table = Hybrid design
Physical designs for diverse workloads

OLTP

B+ tree

Analytics

COL

B+ tree

Are Hybrid Designs important and which workloads can benefit?
Hybrid design = B+ tree & Columnstore
Hybrid design = B+ tree & Columnstore

1. Micro-benchmarks

2. Auto Recommend Hybrid Designs

3. End-to-end evaluation
Micro-benchmarking HYBRID DESIGNS

QUERIES
- Selectivity
- Sort order
- Updates
- Mix: Scans & Updates
- Concurrency

PHYSICAL DESIGNS
- B+ tree
- COL
- B+ tree
Experimental setup for micro-benchmarks

- Synthesized queries and data sets + TPC-H data
- Pre-release version of SQL Server 2017
- Server: on-premise
  - **Dual socket** Intel® Xeon® CPU E5–2660v2, **10 cores per socket**, 2 threads per core, clocked at 2.20 GHz, 64 KB L1 cache per core, 256 KB L2 cache per core, and 25 MB L3 cache shared
  - **384 GB main memory**
  - **18 TB HDD in RAID-0** configuration (throughput of ~1 GB/sec for reads and ~400 MB/sec for writes)
B+ tree Range Scans vs. Col Full Scans

SELECT sum(col1) FROM table WHERE col1 < {1} 
10 GB, 1 int col
B+ tree Range Scans vs. Col Full Scans

SELECT sum(col1) FROM table WHERE col1 < \{1\}
10 GB, 1 int col

Skip data effectively & run single-threaded
B+ tree Range scans vs. Col Full scans

SELECT sum(col1) FROM table WHERE col1 < {1}
10 GB, 1 int col

Superior performance of Columnstore scans
B+ tree Range Scans vs. Col Full Scans

SELECT sum(col1) FROM table WHERE col1 < {1}
10 GB, 1 int col

B+ tree helps for low selectivity & slower storage
B+ tree Range Scans vs. Col Full Scans

```
SELECT sum(col1) FROM table WHERE col1 < {1}
10 GB, 1 int col
```

B+ tree helps for low selectivity & slower storage
SELECT col1, sum(col2) FROM table
GROUP BY col1
20 GB, 2 int col,
*vary number of distinct values in col1*
*B+ tree sorted on col1*
SELECT col1, sum(col2) FROM table
GROUP BY col1
20 GB, 2 int col,
vary number of distinct values in col1
B+ tree sorted on col1

Scanning & hashing Col faster than reading sorted B+ tree
Sort order of B+ tree beneficial for scarce query memory
Two types of Columnstores in SQL Server

Primary Columnstores

- Base table storage
- Optimize large scans
- Complicated updates on compressed columns
- Updates via the delete bitmap are expensive

Compressed row groups  Delete bitmap

Delta stores
Two types of Columnstores in SQL Server

Secondary Columnstores

- Designed to balance performance of long scans and updates
- Delete buffer stores the logical row being deleted
- Requires anti-semi join
Mixed workload: large scans & small updates

**Update** top 10 rows

- Pri. B+ tree
- Pri. B+ tree with Sec. Col
- Pri. Col

**TPC-H 30 GB, 10 concurrent queries, Read Committed**

Hybrid design

Execution time (millisec)

- scan: 0, update: 100
- Percentage (%) for scans and updates
Mixed workload: large scans & small updates

**Update** top 10 rows

TPC-H 30 GB, 10 concurrent queries, Read Committed

Execution time (millisecond)

<table>
<thead>
<tr>
<th>Pri. B+ tree</th>
<th>Pri. B+ tree with Sec. Col</th>
<th>Pri. Col</th>
</tr>
</thead>
</table>
| **B+ trees cheaper than Columnstores for pure updates**

Percentage (%) for scans and updates

scan: 0, update: 100
Mixed workload: large scans & small updates

**Update** top 10 rows, **Select** sum of quantity & price for a single shipdate from lineitem table

---

**TPC-H** 30 GB, 10 concurrent queries, Read Committed

- **Pri. B+ tree**
- **Pri. B+ tree with Sec. Col**
- **Pri. Col**

---

Secondary Columnstore: balance small updates & large scans
Hybrid design = B+ tree & Columnstore

1. Micro-benchmarks

2. Auto Recommend Hybrid Designs

3. End-to-end evaluation
Database Engine Tuning Advisor (DTA)

Workload, Constraints (e.g. storage budget)

Database Server
Query Optimizer
Create Index, Drop Index ...

Output
what-if

DTA
Database Engine Tuning Advisor (DTA)

- **Workload**, Constraints (e.g. storage budget)

- Output

- Database Server
  - Query Optimizer

- Create Index
  - Drop Index
  - ...
Database Engine Tuning Advisor (DTA)

**Workload**, Constraints (e.g. storage budget)

DTA

Database Server

Query Optimizer

Create Index, Drop Index ...

Output

what-if
Database Engine Tuning Advisor (DTA)

Workload, Constraints (e.g. storage budget)

Database Server
Query Optimizer
Create Index
Drop Index ...

what-if
Output
Extensions to SQL Server Query Optimizer

Augment the “what-if” API for:

1. Costing queries on Hypothetical Columnstores
Extensions to SQL Server Query Optimizer

Augment the “what-if” API for:

1. Costing queries on Hypothetical Columnstores

2. Per-column size estimation
   - stay within storage budget
   - per-column access cost
   - hard problem
DTA extensions for hybrid designs

1. Optimal designs searched over:
   combined space of
   Columnstores & B+ trees
DTA extensions for Hybrid Designs

1. Optimal designs searched over:
   combined space of
   Columnstores & B+ trees

2. Released as part of
   SQL Server 2017 CTP
Hybrid design = B+ tree & Columnstore

1. Micro-benchmarks

2. Auto Recommend Hybrid Designs

3. End-to-end evaluation
End-to-end evaluation of hybrid designs

- 5 customer workloads, TPC-DS, and CH benchmark
- Homogenous design
  - B+ tree only (for each query, tune with DTA’s B+ tree index recommendations)
  - Columnstore only (secondary Columnstores on all tables for all possible columns)
- Hybrid design
  - B+ tree and Columnstore indexes recommended by DTA (Database Engine Tuning Advisor)
- Speedup = (time on Homogenous) / (time on Hybrid)
  - Metrics: median execution time & CPU time (per query)
  - Warm execution & working set in memory
- The same hardware & software as for micro-benchmarks
Hybrid Query Plans are common

![Graph showing the percentage of hybrid query plans for different customers and TPC-DS benchmark.]

- **Cust1**: Approximately 80% hybrid query plans
- **Cust2**: Approximately 20% hybrid query plans
- **Cust3**: Approximately 40% hybrid query plans
- **Cust4**: Approximately 100% hybrid query plans
- **Cust5**: Approximately 50% hybrid query plans
- **TPC-DS**: Approximately 40% hybrid query plans

The red bars represent the percentage of hybrid query plans for each customer and TPC-DS benchmark.
Hybrid Query Plans are common

Percentage of query plans

- Cust1
- Cust2
- Cust3
- Cust4
- Cust5
- TPC-DS

Hybrid Query Plans
Hybrid Query Plans (Same Table)
Columnstores & B+ trees selected by DTA

TPC-DS benchmark
100 GB

# of Queries

Bins of Speedup (CPU time)

- Hybrid Vs B+ tree only
Columnstores & B+ trees selected by DTA

TPC-DS benchmark
100 GB

Hybrid Vs B+ tree only

Bins of Speedup (CPU time)

<table>
<thead>
<tr>
<th>Speedup</th>
<th># of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>7</td>
</tr>
<tr>
<td>0.8</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>10</td>
</tr>
<tr>
<td>1.5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>&gt;2</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speedup</th>
<th># of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>&gt;2</td>
<td>23</td>
</tr>
</tbody>
</table>

37
Columnstores & B+ trees selected by DTA

TPC-DS benchmark
100 GB

Bins of Speedup (CPU time)

- Hybrid Vs B+ tree only
- Hybrid Vs Columnstore only

# of Queries

- 0.5: 7 (Hybrid), 4 (Columnstore)
- 0.8: 5 (Hybrid), 16 (Columnstore)
- 1.2: 10 (Hybrid), 46 (Columnstore)
- 1.5: 5 (Hybrid), 11 (Columnstore)
- 2: 10 (Hybrid), 4 (Columnstore)
- 5: 27 (Hybrid), 4 (Columnstore)
- 10: 23 (Hybrid)
- >10: 10 (Hybrid), 11 (Columnstore)
Hybrid designs significantly improve decision support workload
Example: Hybrid Design for TPC-DS Q54

- Q54: several selective predicates on dimension table: date_dim
- DTA: B+ tree index on date_dim as well as fact tables catalog_sales and web_sales
- 20X reduction in execution cost of leaf nodes (8,000 ms to 400 ms),
- 10X reduction in query execution cost (from 26,000 ms to 2,600 ms)
CH Benchmark: OLTP + Analytics

15 cores for analysis queries, 5 cores for transactions, workload runs 6 hours, 1K warehouses

No. of Queries vs Bins of Speedup (wall-clock time):

- Hybrid Vs B+ tree only (Snapshot Isolation)

<table>
<thead>
<tr>
<th>Bins of Speedup (wall-clock time)</th>
<th>0.5</th>
<th>0.8</th>
<th>1.2</th>
<th>1.5</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>&gt;10</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Queries</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

41
CH Benchmark: OLTP + Analytics

15 cores for analysis queries, 5 cores for transactions, workload runs 6 hours, 1K warehouses

No. of Queries

Hybrid Vs B+ tree only (Snapshot Isolation)

Bins of Speedup (wall-clock time)

2X slowdown of transactions & 10X speed-up of analytics
Summary
Hybrid Designs: crucial for mixed workloads

- Hybrid physical designs should not be ignored!
  - Effective even for read-only queries

- Small slowdown for OLTP transactions and 
  10X or more speed-up of complex analytical queries

- DTA can recommend combination of Columnstores and 
  B+ trees

- Several open challenges
  - Columnstore size estimation
  - Modeling concurrency effects
Thank you
Notes

- Primary index – means that this is the main storage of the data
- Mixed workload insight: as soon as we have one big scan, the Hybrid Design provides much better performance
- Primary Columnstores not efficient for mixed workloads
- DTA – optimization tool + access to the query optimizer via the what-if API
- SQL Server Columnstore not globally ordered
- Explore when to use B+ tree or a Columnstore
- SQL supports hybrid designs (not new, already there)
Tease apart performance characteristics

- Vectorized
- Compressed
- Late materialization
- Sorted globally
- Fast modifications
- Memory efficient

B+ tree

Selectivity values
Sortedness
Concurrency
Takeaways from micro-benchmark analysis

- point lookups & short scans
- balance scans & updates
- large & fast scans
- streaming via sortedness
- Leverage both B+ tree & Col
- Batch-mode & compression
Key takeaways from end-to-end evaluation

- Hybrid designs can result in 10X to 100X improvement in execution costs compared to B+ tree or Columnstore only designs
- DTA – automated recommendation of hybrid physical designs is cost-based and workload-dependent
- Open challenges for hybrid designs in query optimization, concurrency, and locking
Compression with RLE & GEE estimator

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.0</td>
</tr>
<tr>
<td>30</td>
<td>0.1</td>
</tr>
<tr>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
</tr>
<tr>
<td>30</td>
<td>0.0</td>
</tr>
<tr>
<td>30</td>
<td>0.1</td>
</tr>
<tr>
<td>30</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
</tr>
<tr>
<td>1, 1</td>
</tr>
<tr>
<td>3, 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 3</td>
</tr>
<tr>
<td>1, 3</td>
</tr>
</tbody>
</table>

Initial sample  Encoded  Sorted by B,A  Compressed

**GEE estimator** groups that occur once in the sample are scaled by \( \frac{\text{total}_\text{size}}{\text{sample}_\text{size}} \) (e.g. \([0,1]\) and \([1,1]\)), other groups are counted once in total.
Columnstore size estimation

- Not scalable to scan and apply encodings on the whole dataset
- Estimate per column/segment sizes using sampling
- For each column segment: 1) encode values 2) determine optimal row ordering 3) compress.
- Variety of encodings, row ordering optimization and compression techniques make size estimation hard
## Customer workloads and benchmarks

<table>
<thead>
<tr>
<th>Workload</th>
<th>DB Size (GB)</th>
<th># of tables</th>
<th>Max table size (GB)</th>
<th>Avg. # of cols per table</th>
<th># of queries</th>
<th>Avg. # of joins</th>
<th>Avg. # of ops per plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust1</td>
<td>172</td>
<td>23</td>
<td>63.8</td>
<td>14.1</td>
<td>36</td>
<td>7.2</td>
<td>29.1</td>
</tr>
<tr>
<td>Cust2</td>
<td>44.6</td>
<td>614</td>
<td>44.6</td>
<td>23.5</td>
<td>40</td>
<td>8.1</td>
<td>28.3</td>
</tr>
<tr>
<td>Cust3</td>
<td>138.4</td>
<td>3394</td>
<td>79.8</td>
<td><strong>26.3</strong></td>
<td>40</td>
<td>8.7</td>
<td>24.1</td>
</tr>
<tr>
<td>Cust4</td>
<td>93</td>
<td>22</td>
<td>54.8</td>
<td>20.3</td>
<td>24</td>
<td>6.9</td>
<td>24.4</td>
</tr>
<tr>
<td>Cust5</td>
<td>9.83</td>
<td>474</td>
<td>1.52</td>
<td>5.5</td>
<td>47</td>
<td><strong>23.1</strong></td>
<td><strong>57.7</strong></td>
</tr>
<tr>
<td>TPC-DS</td>
<td>87.7</td>
<td>24</td>
<td>34.9</td>
<td>17.2</td>
<td><strong>97</strong></td>
<td>7.9</td>
<td>28.2</td>
</tr>
<tr>
<td>TPC-CH</td>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivation

- Wide variety of workloads with different usage characteristics: OTLP, OLAP, Mixed workloads (e.g. real-time analytics)
- Mixed workloads imply need for Hybrid Physical Designs to achieve good performance
- SQL Server supports a wide variety of Physical Design options
- What are the ramifications of Hybrid Designs for auto-indexing in SQL Server?
Major Questions

- Efficient data skipping and sort order of B+ tree vs. efficient vectorized processing of Columnstores
  - What are the trade-offs?
    - Data skipping
    - Concurrency
    - Memory constraints

- Impact of updates on B+ trees & Columnstores
  - Primary (clustered) and secondary (non-clustered) have different designs

- Performance of Hybrid Designs for Mixed Workloads
  - short updates and reporting queries scanning data
Two types of Columnstore Indexes (CSIs)

- CSI updatable via auxiliary structures
- Primary CSI designed to optimize scan performance in DW
  - No delete buffer in primary CSI, making small updates expensive
Characteristics: B+ tree, Col & hybrid

- SORTED B+ TREE
- PRIMARY B+ TREE
- &
- SECONDARY COLUMNSTORE

- NOT SORTED GLOBALLY COLUMNSTORE
- COLUMNSTORE PRIMARY STORAGE
- &
- SECONDARY B+ TREE
Takeaways from micro-benchmark analysis

- B+ tree
  - short range scans & lookups
  - large scans & bulk updates
  - Balance updates & scans
- Scarcce memory
- Mixed workload
- Small storage footprint

COL
Takeaways from Microbenchmark analysis

- **B+ trees**: suitable for point lookups, short range scans, update-only
- Sortedness of B+ trees only help when memory is insufficient
- **Secondary CSIs** strike a right balance between scans and updates
- **Hybrid physical designs** are essential for many mixed workloads where updates often have selective predicates
Mixed Workload: CH BenCHmark

- A join between TPC-C benchmark with the analytical H component from TPC-H
- C component: unmodified TPC-C schema and 5 transactions
- H component: 3 additional tables and adapted 22 queries which mimic TPC-H queries
- Concurrent queries compete for resources & locks
- Affinitize component C & H to different cores
Why is Columnstore size estimation hard?

Large data & design space

Complex storage

Sample based techniques

1) Build index on samples
2) Model full index
Hypothetical Columnstore size estimation

Build index on samples

+ Simple
+ No changes required when index storage modified
- Low accuracy
- High overhead (run algorithms and store index)

Model full index

+ More accurate
+ No overhead of sorting or writing index to disk
- High maintenance cost
- Complex (uses GEE estimator)
Hypothetical Columnstore size estimation

- **Correctness:** do not go over storage budget
- **Efficiency:** cannot afford to build the whole index
- **Accuracy:**
  - estimate the size on samples of data
  - model complex storage involving encoding, sorting and compression
Update performance

UPDATE N rows lineitem
WHERE l_shipdate = {1}
TPC-H 30 GB, single thread

Primary & Secondary Columnstores comparable for large updates
Hybrid Query Plans are common

Percentage of queries

Cust1 | Cust2 | Cust3 | Cust4 | Cust5 | TPC-DS

Hybrid Plans (Same Query) | Hybrid Plans (Same Table)

0% | 20% | 40% | 60% | 80% | 100%
Update performance

Primary Columnstores incur high cost for small updates
Cheaper updates for B+ trees than for Columnstores
Why is Columnstore size estimation hard?

Large data & design space

Sample based techniques

Complex Columnar storage

At least as hard as distinct value estimation

Input | Encode | Sort  | Compress
---|---|---|---
A 0.03 | A 3 | A 2 | A 2, 3
A 0.02 | A 2 | A 2 | A 3, 2
A 0.02 | A 2 | A 2 | A 3
A 0.03 | A 3 | A 3 | A 3
Hybrid Query Plans are common

Percentage of query plans

Cust1 | Cust2 | Cust3 | Cust4 | Cust5 | TPC-DS

Hybrid Query Plans
Hybrid Query Plans (Same Table)
Why is Columnstore size estimation hard?

Large data & design space

Sample based techniques

Complex Columnar storage

At least as hard as distinct value estimation
Mixed workload: large scans & small updates

*Update* top 10 rows, *Select* sum of quantity & price for a single shipdate from lineitem table

- **Pri. B+ tree**
- **Pri. B+ tree with Sec. Col**
- **Pri. Col**

<table>
<thead>
<tr>
<th>Execution time (millisecond)</th>
<th>Percentage (%) for scans and updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan: 0, update: 100, update: 99</td>
<td>72%</td>
</tr>
<tr>
<td>scan: 1, update: 99</td>
<td>72%</td>
</tr>
<tr>
<td>scan: 2, update: 98</td>
<td>72%</td>
</tr>
<tr>
<td>scan: 3, update: 97</td>
<td>72%</td>
</tr>
<tr>
<td>scan: 4, update: 96</td>
<td>72%</td>
</tr>
<tr>
<td>scan: 5, update: 95</td>
<td>72%</td>
</tr>
</tbody>
</table>

Secondary Columnstore: balance small updates & large scans
<table>
<thead>
<tr>
<th>Concept</th>
<th>Access Path Selection</th>
<th>Hybrid designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>B+ tree</td>
<td>Main memory optimized</td>
<td>General (disk-based)</td>
</tr>
<tr>
<td>Scans</td>
<td>Shared</td>
<td>Non-shared</td>
</tr>
<tr>
<td>DB engine</td>
<td>Prototype</td>
<td>SQL Server</td>
</tr>
<tr>
<td>Main focus</td>
<td>Model Concurrency</td>
<td>DTA and many workloads</td>
</tr>
<tr>
<td>Physical designs</td>
<td>Columnstore and Secondary B+tree</td>
<td>Hybrid Physical Designs</td>
</tr>
</tbody>
</table>
Hybrid Designs

Primary B+ tree (base table)
Secondary Columnstore
*Secondary B+tree on ship date*

Heap file base table
Secondary Columnstore
*Secondary B+tree on ship date*

B+ tree clustered on order & linenumber
Covers all columns

Heap file
Covers all columns
Hybrid Designs

Primary B+ tree (base table)
Secondary Columnstore
Secondary B+ tree on ship date

Primary Columnstore
Secondary B+ tree on order/line
Secondary B+ tree on ship date

Primary B+ tree (base table)
Secondary Columnstore
Secondary B+ tree on ship date

Covers all columns

B+ tree clustered on order & linenumber

Ship date
Database Engine Tuning Advisor (DTA)

**Workload**

Constraints - Storage budget - ...

1) Select candidates
2) Merge Indexes
3) Enumerate

**COST**

**what-if**

DB Engine
- Query Optimizer

Apply

Create Index
Drop Index
Create View ...

Output
Database Engine Tuning Advisor (DTA)

**Workload**
Constraints - Storage budget - ...

1) Select candidates
2) Merge Indexes
3) Enumerate

**DTA**

**COST**

what-if

**DB Engine**
Query Optimizer

Create Index
Drop Index
Create View ...

Output

Apply
Database Engine Tuning Advisor (DTA)

- Workload
  - Constraints
    - Storage budget
    - ...

1) Select candidates
2) Merge Indexes
3) Enumerate

C O S T

what-if

DB Engine
Query Optimizer

Apply

Create Index
Drop Index
Create View
...

Output
Database Engine Tuning Advisor (DTA)

Workload
Constraints - Storage budget - ...

1) Select candidates
2) Merge Indexes
3) Enumerate

COST

what-if

DB Engine
Query Optimizer

Apply

Create Index
Drop Index
Create View
...

Output
Primary Columnstore & Secondary B+ tree index:
- Efficient point-lookups and small range scans \w B+ tree
- enforce constraints efficiently (e.g. Primary Key)

Source: Real-Time Analytical Processing with SQL Server – Paul Larson, Adrian Birka, Eric Hanson, Weiyun Huang, Michal Novakiewicz, Vassilis Papadimos (Microsoft)