Touchstone: Generating Enormous Query-Aware Test Databases

Yuming Li\textsuperscript{1}, Rong Zhang\textsuperscript{1}, Xiaoyan Yang\textsuperscript{2}, Zhenjie Zhang\textsuperscript{2}, Aoying Zhou\textsuperscript{1}

\textsuperscript{1}DaSE at East China Normal University
\textsuperscript{2}Singapore R&D, Yitu Technology Ltd.
Test Databases Are Important!

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Test Databases Are Important!

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Test Databases Are Important!

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Test Databases Are Important!

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Test Databases Are Important!

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Test Databases Are Important!

Application scenarios:
- DBMS testing
- Database application testing
- Application-driven benchmarking

IT company 
Solution provider

Production DB

Bank Application

1. Slow!
2. It is optimized, and test is OK!
3. Still slow!!

What? Why?

- Application scenarios: DBMS testing, database application testing, application-driven benchmarking.
Random Test Database Is Deficient!

- The random test database has the **same** database schema and data characteristics as database generated by dbgen.
- There are **huge** execution cost differences between realistic database (dbgen) and synthetic database (random).

Comparing the query latencies over database generated by dbgen and database randomly generated.

The average relative error of query latencies is **175%!!**
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.
Query-Aware Data Generation

- **Input**: database schema, data characteristics and workload characteristics.
- **Output**: test database and instantiated query parameters.
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.

### Database Schema $H$

<table>
<thead>
<tr>
<th>$R$ size: 10</th>
<th>$S$ size: 20</th>
<th>$T$ size: 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$ Primary Key</td>
<td>$s_1$ Primary Key</td>
<td>$t_1$ Primary Key</td>
</tr>
<tr>
<td>$r_2$ Integer</td>
<td>$s_1$ Foreign Key($r_1$)</td>
<td>$t_1$ Foreign Key($s_1$)</td>
</tr>
<tr>
<td>$r_3$ Double</td>
<td>$s_1$ Integer</td>
<td>$t_1$ Varchar</td>
</tr>
</tbody>
</table>

### Data Characteristics $D$

<table>
<thead>
<tr>
<th>Attr</th>
<th>Null</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R.r_1$</td>
<td>0</td>
<td>[0, 10]</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>$R.r_2$</td>
<td>0</td>
<td>[0, 30]</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$S.s_1$</td>
<td>0</td>
<td>[0, 20]</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>$T.t_1$</td>
<td>20%</td>
<td>--</td>
<td>8</td>
<td>(20, 100)</td>
</tr>
</tbody>
</table>
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.

### Database Schema $H$

<table>
<thead>
<tr>
<th>R</th>
<th>Attr</th>
<th>Null</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$r_2$</td>
<td>$r_3$</td>
<td></td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>$r_1$</td>
<td>$r_3$</td>
<td>$s_1$</td>
<td>0,10</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_3$</td>
<td>$t_1$</td>
<td>0,30</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_3$</td>
<td>$t_3$</td>
<td>0,20</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$t_3$</td>
<td></td>
<td>20%</td>
<td>8</td>
<td>(20,100)</td>
</tr>
</tbody>
</table>

### Data Characteristics $D$

### Workload Characteristics $W$

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[Q_1, R.r_3 &lt; P_1, 4]$</td>
<td>$[Q_2, S.s_3 = P_5, 5]$</td>
</tr>
<tr>
<td>$[Q_1, S.s_3 = P_2, 3]$</td>
<td>$[Q_1, R.r_1 = S.s_3, 3]$</td>
</tr>
<tr>
<td>$[Q_1, R.r_1 = S.s_2, 2]$</td>
<td>$[Q_2, T.t_3 = (P_5, P_6), 12]$</td>
</tr>
<tr>
<td>$[Q_2, S.s_3 &gt; P_9, 7]$</td>
<td>$[Q_2, S.s_3 = T.t_3, 7]$</td>
</tr>
<tr>
<td>$[Q_2, T.t_3 NOT LIKE P_6, 32]$</td>
<td>$[Q_2, 2 \times R.r_1 + R.r_3 &lt; P_9, 7]$</td>
</tr>
<tr>
<td>$[Q_2, S.s_3 = T.t_3, 5]$</td>
<td>$[Q_2, S.s_3 = P_9, 12]$</td>
</tr>
<tr>
<td>$[Q_1, R.r_2 - R.r_3 &gt; P_9, 6]$</td>
<td>$[Q_1, S.s_3 = R.r_2, 11, 53]$</td>
</tr>
</tbody>
</table>

### Query Execution Trees

- **Parameterized queries**
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.

### Database Schema $H$

<table>
<thead>
<tr>
<th>Database</th>
<th>$R$ size: 10</th>
<th>$S$ size: 20</th>
<th>$T$ size: 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>Primary Key</td>
<td>$s_1$</td>
<td>Primary Key</td>
</tr>
<tr>
<td>$r_2$</td>
<td>Integer</td>
<td>$s_1$</td>
<td>Foreign Key</td>
</tr>
<tr>
<td>$r_3$</td>
<td>Double</td>
<td>$s_1$</td>
<td>Integer</td>
</tr>
<tr>
<td>$t_1$</td>
<td>Primary Key</td>
<td>$t_1$</td>
<td>Foreign Key</td>
</tr>
<tr>
<td>$t_2$</td>
<td>Foreign Key</td>
<td>$t_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$t_3$</td>
<td></td>
<td>$t_1$</td>
<td>Varchar</td>
</tr>
</tbody>
</table>

### Data Characteristics $D$

<table>
<thead>
<tr>
<th>Attr</th>
<th>Null</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>0</td>
<td>[0, 10]</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>$r_2$</td>
<td>0</td>
<td>[0, 30]</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$s_1$</td>
<td>0</td>
<td>[0, 20]</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>$t_3$</td>
<td>20%</td>
<td></td>
<td>8</td>
<td>(20, 100)</td>
</tr>
</tbody>
</table>

### Workload Characteristics $W$

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$[Q_2, R_1 &lt; P_1, 4]$</th>
<th>$C_8$</th>
<th>$[Q_2, S_5 = P_6, 5]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_2$</td>
<td>$[Q_2, S_5 = P_2, 3]$</td>
<td>$C_9$</td>
<td>$[Q_1, R_1 = S_5, 3]$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$[Q_2, R_1 = S_5, 2]$</td>
<td>$C_{10}$</td>
<td>$[Q_1, T_3 \text{ IN} (P_5, P_6), 12]$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$[Q_2, S_5 \geq P_7, 7]$</td>
<td>$C_{11}$</td>
<td>$[Q_1, S_5 = T_5, 7]$</td>
</tr>
<tr>
<td>$C_5$</td>
<td>$[Q_2, T_3 \text{ NOT LIKE} P_6, 32]$</td>
<td>$C_{12}$</td>
<td>$[Q_2, 2 \times R_1 + R_1 &lt; P_5, 7]$</td>
</tr>
<tr>
<td>$C_6$</td>
<td>$[Q_2, S_5 = T_3, 5]$</td>
<td>$C_{13}$</td>
<td>$[Q_2, S_5 \geq P_2, 12]$</td>
</tr>
<tr>
<td>$C_7$</td>
<td>$[Q_2, R_2 &gt; R_1 &gt; P_6, 6]$</td>
<td>$C_{14}$</td>
<td>$[Q_2, S_5 &lt; R_1 \geq P_11, 53]$</td>
</tr>
</tbody>
</table>

### Query Execution Trees

- **Query 1:** $Q_1: R \rightarrow S \rightarrow T$
  - $Q_2: \sum_{S_5}$
  - $T_{t_3}$

- **Query 2:** $Q_3: S \rightarrow S$
  - $S_5$
  - $P_3$

- **Query 3:** $Q_4: R \rightarrow S$
  - $R_2$
  - $R_3$
  - $P_5$

- **Variable parameters**
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.

### Input
- **Database Schema $H$**
  - $R_1$: Primary Key
  - $R_2$: Integer
  - $R_3$: Double
  - $S_1$: Primary Key
  - $S_2$: Foreign Key ($R_1$)
  - $S_3$: Integer

- **$T_1$:**
  - Primary Key
  - $T_2$: Foreign Key ($S_3$)
  - $T_3$: Varchar

### Output
- **Data Characteristics $D$**

<table>
<thead>
<tr>
<th>Attr</th>
<th>Null</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>[0, 10]</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>$R_2$</td>
<td>0</td>
<td>[0, 30]</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$S_1$</td>
<td>0</td>
<td>[0, 20]</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>$T_3$</td>
<td>20%</td>
<td>--</td>
<td>8</td>
<td>(20, 100)</td>
</tr>
</tbody>
</table>

### Workload Characteristics $W$

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$[Q_1, R_3 &lt; P_1, 4]$</th>
<th>$C_8$</th>
<th>$[Q_2, S_3 = P_5, 5]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_2$</td>
<td>$[Q_1, S_3 - S_2, 3]$</td>
<td>$C_9$</td>
<td>$[Q_1, R_1 - S_3, 3]$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$[Q_1, R_1 = S_3, 2]$</td>
<td>$C_{10}$</td>
<td>$[Q_1, T_3 \text{ IN (P_5, P_6)}, 12]$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$[Q_2, S_3 = P_7, 7]$</td>
<td>$C_{11}$</td>
<td>$[Q_1, S_3 - T_3, 7]$</td>
</tr>
<tr>
<td>$C_5$</td>
<td>$[Q_2, T_3 \text{ NOT LIKE P}_6, 32]$</td>
<td>$C_{12}$</td>
<td>$[Q_2, 2 \times R_3 + R_5 &lt; P_6, 7]$</td>
</tr>
<tr>
<td>$C_6$</td>
<td>$[Q_2, S_3 - T_3, 5]$</td>
<td>$C_{13}$</td>
<td>$[Q_2, S_3^2 = P_1, 12]$</td>
</tr>
<tr>
<td>$C_7$</td>
<td>$[Q_2, R_2 - R_3 &gt; P_9, 6]$</td>
<td>$C_{14}$</td>
<td>$[Q_4, S_3 - R_3 \geq P_1, 53]$</td>
</tr>
</tbody>
</table>

### Query Execution Trees

1. $Q_1: \quad \gamma_{T_3 \text{, } S_1 \text{, } T_2} \text{ SUM(S}_3\text{)} = 5$

2. $Q_2: \quad \gamma_{T_3 \text{, } S_1 \text{, } T_2} \text{ SUM(S}_3\text{)} = 5$

3. $Q_3: \quad \gamma_{R_2 \text{, } T_3 \text{, } R_1 \text{, } S_3 \text{, } T_1} \text{ SUM(S}_3\text{)} = 5$

4. $Q_4: \quad \gamma_{R_2 \text{, } R_3 \text{, } P_9 \text{, } S_3 \text{, } R_5 \text{, } P_10} \text{ SUM(S}_3\text{)} = 5$

Cardinality constraints
Query-Aware Data Generation

**Input:** database schema, data characteristics and workload characteristics.

**Output:** test database and instantiated query parameters.

<table>
<thead>
<tr>
<th>R size: 10</th>
<th>S size: 20</th>
<th>T size: 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( s_1 )</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>Primary Key</td>
<td>Primary Key</td>
<td>Primary Key</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>( s_1 )</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>Integer</td>
<td>Foreign Key(( r_2, r_3 ))</td>
<td>Foreign Key(( s_1, s_3 ))</td>
</tr>
<tr>
<td>( r_3 )</td>
<td>( s_1 )</td>
<td>Varchar</td>
</tr>
</tbody>
</table>

**Data Characteristics**

<table>
<thead>
<tr>
<th>Attr</th>
<th>Null</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R, r_2 )</td>
<td>0</td>
<td>([0, 10])</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>( R, r_3 )</td>
<td>0</td>
<td>([0, 30])</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>( S, s_3 )</td>
<td>0</td>
<td>([0, 20])</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>( T, t_3 )</td>
<td>20%</td>
<td>--</td>
<td>8</td>
<td>((20, 100))</td>
</tr>
</tbody>
</table>

**Workload Characteristics**

- \( C_1 \): \( |Q_1, R, r_3 < P_1, 4| \)
- \( C_2 \): \( |Q_1, S, s_3 = P_2, 3| \)
- \( C_3 \): \( |Q_1, R, r_1 = S, s_2, 2| \)
- \( C_4 \): \( |Q_2, S, s_3 > P_3, 7| \)
- \( C_5 \): \( |Q_2, T, t_3 NOT LIKE P_4, 32| \)
- \( C_6 \): \( |Q_2, S, s_1 = T, t_3, 5| \)
- \( C_7 \): \( |Q_3, R, r_2 = R, r_3 > P_5, 6| \)
- \( C_8 \): \( |Q_3, S, s_3 = P_6, 5| \)
- \( C_9 \): \( |Q_4, R, r_1 = S, s_2, 3| \)
- \( C_{10} \): \( |Q_4, T, t_3 IN (P_5, P_6), 12| \)
- \( C_{11} \): \( |Q_5, S, s_1 = T, t_3, 7| \)
- \( C_{12} \): \( |Q_5, 2 \times R, r_3 < P_6, 7| \)
- \( C_{13} \): \( |Q_5, S, s_3 = P_6, 12| \)
- \( C_{14} \): \( |Q_5, S, s_3 = R, r_3 > P_5, 53| \)

All 14 cardinality constraints
Query-Aware Data Generation

- **Input:** database schema, data characteristics and workload characteristics.
- **Output:** test database and instantiated query parameters.

**Cardinality constraints on Q₁**
Query-Aware Data Generation

**Input:** database schema, data characteristics and workload characteristics.

**Output:** test database and instantiated query parameters.

### Database Schema $\mathcal{H}$

<table>
<thead>
<tr>
<th>Attr</th>
<th>Domain</th>
<th>Cardinality</th>
<th>Len(Avg,Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R.r_2$</td>
<td>$[0, 10]$</td>
<td>5</td>
<td>--</td>
</tr>
<tr>
<td>$R.r_3$</td>
<td>$[0, 30]$</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>$S.s_3$</td>
<td>$[0, 20]$</td>
<td>10</td>
<td>--</td>
</tr>
<tr>
<td>$T.t_3$</td>
<td>20%</td>
<td>8</td>
<td>(20, 100)</td>
</tr>
</tbody>
</table>

### Data Characteristics $\mathcal{D}$

### Workload Characteristics $\mathcal{W}$

- $C_1: [Q_1, R.r_3 < P_1, 4]$
- $C_2: [Q_1, S.s_3 = P_2, 3]$
- $C_3: [Q_1, R.r_1 = S.s_2, 2]$
- $C_4: [Q_2, S.s_3 > P_3, 7]$
- $C_5: [Q_1, T.t_3 \text{ NOT LIKE } P_4, 12]$
- $C_6: [Q_2, S.s_1 = T.t_2, 5]$
- $C_7: [Q_1, R.r_2 - R.r_3 > P_5, 6]$

- $C_8: [Q_2, S.s_3 = P_5, 5]$
- $C_9: [Q_1, R.r_1 = S.s_2, 3]$
- $C_{10}: [Q_2, T.t_3 \text{ IN } (P_5, P_6), 12]$
- $C_{11}: [Q_2, S.s_3 > P_3, 7]$
- $C_{12}: [Q_1, 2 \times R.r_2 + R.r_3 < P_5, 12]$
- $C_{13}: [Q_1, S.s_3 > P_5, 12]$
- $C_{14}: [Q_1, S.s_3 - R.r_3 \geq P_11, 53]$

### Query Execution Trees

- $Q_1$: $\text{size} = 2$
- $Q_2$: $\text{size} = 3$
- $Q_3$: $\text{size} = 7$
- $Q_4$: $\text{size} = 12$

### Output

- $Q_1$: $\sum_{T.t_3} \text{sum}(S.s_3)$
- $Q_2$: $\text{size} = 5$
- $Q_3$: $\text{size} = 12$
- $Q_4$: $\text{size} = 53$
Comparison to Related Works

The performance of state-of-the-art solutions remains far from satisfactory.

<table>
<thead>
<tr>
<th></th>
<th>QAGen SIGMOD 2007</th>
<th>WAGen VLDB 2010</th>
<th>DCGen SIGMOD 2011</th>
<th>MyBenchmark VLDBJ 2014</th>
<th>Touchstone ATC 2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Parallelization</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Scalability</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Austere Mem Consumption</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Wide Workload Support</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Minimal Human Effort</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Touchstone is the first query-aware data generator which can support full parallel data generation on multiple nodes. And Touchstone is capable of supporting industrial scale database generation.
Why do Previous Studies not Work?

- **Primitive data generation algorithm**
  - Can not support fully parallel data generation in a distributed environment;
  - Can not support the non-equi-join workload.

- **Huge intermediate state dataset**
  - The memory consumption strongly depends on the size of generation outputs;
  - Is not scalable in generation database size.
How does Touchstone Solve These Problems?

- **New query instantiation scheme**
  - Algorithms: binary search, random sampling;
  - Function: instantiating all the variable parameters.

- **New data generation scheme**
  - Algorithms: data generation using constraint chains, data compression on join information table;
  - Function: facilitating parallel data generation on multiple nodes with austere memory consumption.
Overall Architecture

**Query instantiation**
- Initialize random column generators;
- Instantiate symbolic query parameters.

**Data generation**
- Decompose the query trees annotated with cardinality constraints into constraint chains;
- Generate data in parallel on multiple nodes.
Overall Architecture

- **Query instantiation**
  - Initialize random column generators;
  - Instantiate symbolic query parameters.

- **Data generation**
  - Decompose the query trees annotated with cardinality constraints into constraint chains;
  - Generate data in parallel on multiple nodes.

### Diagram

- **Input**
  - $H$: $r_1$: Primary Key, $r_2$: Integer, $r_3$: Double
  - $D$: $R_r_2$: 0, $R_r_3$: 0, $[0, 10]$

- **Query Instantiation**
  - **Init Random Column Generators**
  - **Query Instantiatior**
    - Equality constraints over filters
      - $S_s_3 = P_2$
      - $T.t_3 \text{ NOT LIKE } P_4$
    - Non-Equality constraints over filters
      - $R_r_2 < P_1$
      - $R_r_2 - R_r_3 > P_5$
    - Non-Equality constraints over joins
      - $S.s_3 - R.r_3 \geq P_{11}$

- **Determine the data distribution of columns**
Overall Architecture

### Query instantiation
- Initialize random column generators;
- Instantiate symbolic query parameters.

### Data generation
- Decompose the query trees annotated with cardinality constraints into constraint chains;
- Generate data in parallel on multiple nodes.

---

**Compilation Step**

1. Order the tables as a generation sequence
2. Decompose the query trees into constraint chains

**Assembling Step**

1. Generate tuples in parallel on multiple nodes
2. Efficiently manage the join information
Experiments

- **Test environment**
  - Cluster: 8 nodes
  - CPU: 2 * Intel Xeon E5-2620 @ 2.0 GHz
  - DRAM: 64GB
  - Disk: 3TB HDD configured in RAID-5
  - Network: 1 Gigabit Ethernet

- **Test workloads**
  - TPC-H benchmark (the first 16 queries) & Star schema benchmark (all 13 queries)

- **Comparison**
  - MyBenchmark [VLDBJ 2014]
Touchstone outperforms MyBenchmark on data generation throughput by orders of magnitude.
The memory consumption of Touchstone is minimal.
Touchstone has linear scalability
The workload on synthetic database matches the expectation on result cardinality and query latency.
Limitations & Conclusion

**Limitations:**

- Touchstone does not support filters on key columns;
- Equality constraints over filters involving multiple columns are not supported;
- Equi-joins on columns with no reference constraint are not supported;
- Touchstone does not support the database schema with cyclic reference relationship.

**Conclusion:**

- Touchstone is a query-aware data generator with characteristics of **completely parallelizable** and **bounded usage to memory**. And Touchstone is **linearly scalable to computing resource and data scale**.
Thank you!! Q & A

Test Databases Are Important!

- Applications: DBMS testing, database application testing, application-driven benchmarking.
Random Column Generator

- **Random index generator** outputs indexes from 0 to n-1 while n is the specified **cardinality**, and manipulates the **data distribution** of column values.

- **Index2Value transformer** **deterministically** maps the index to a **concrete value** in the specified domain of the column.

Data Characteristics of Column T.t₃:
Null: 20%, Cardinality: 8, Length: (Avg:20, Max:100)

<table>
<thead>
<tr>
<th>Index</th>
<th>Probability</th>
<th>CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20%</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>15%</td>
<td>0.65</td>
</tr>
<tr>
<td>6</td>
<td>15%</td>
<td>0.9</td>
</tr>
</tbody>
</table>

value = index + Seeds[index mod 3]
Seeds: {ranStr1, ranStr2, ranStr3}
Query Instantiation

The query instantiation is responsible for handling three types of cardinality constraints, i.e., $C_{\leq}, C_{\neq}, C_{\neq \bowtie}$. The fourth type of constraints $C_{\bowtie}$ is taken care of by the data generation process at runtime.

This is an iterative process!

<table>
<thead>
<tr>
<th>Equality constraints over filters</th>
<th>Non-Equality constraints over filters</th>
<th>Non-Equality constraints over joins</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S.s_3 = P_2$</td>
<td>$R.r_2 &lt; P_1$</td>
<td>$S.s_3 - R.r_3 \geq P_{11}$</td>
</tr>
<tr>
<td>T.t_3 NOT LIKE P_4</td>
<td>$R.r_2 - R.r_3 &gt; P_5$</td>
<td>Search a value for $P_{11}$</td>
</tr>
<tr>
<td>Assign a value to $P_2/P_4$ and adjust the data distribution</td>
<td>Search a value for $P_1/P_5$ based on the fixed data distribution</td>
<td>after settling the child nodes</td>
</tr>
</tbody>
</table>
Equality Constraints over Filters

- **(1)** Randomly select an index and obtain the corresponding value for instantiating the parameter;

- **(2)** Update the occurrence probability of the selected index in the column generator;

- **(3)** Calculate the cumulative probabilities in the probability table.

<table>
<thead>
<tr>
<th>Index</th>
<th>Probability</th>
<th>CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20%</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>15%</td>
<td>0.65</td>
</tr>
<tr>
<td>6</td>
<td>15%</td>
<td>0.9</td>
</tr>
</tbody>
</table>

\[ \frac{40 - 32}{40} = 20\% \]

\[ \frac{6}{((1-20\%)*50)} = 15\% \]

\[ ((1 - 20\%)*50) \]

\[ 6 / ((1 - 20\%)*50) = 15\% \]
Non-Equality Constraints over Filters

- Run a **binary search** over the parameter domain to find the optimal concrete parameter based on the fixed column data distribution.

- Using the **random sampling** algorithm to evaluate the probability of tuples satisfying the instantiated predicate.

Parameter searching procedure

$C_7 = [Q_3, R.r_2 - R.r_3 > P_5, 6]$
Non-Equality Constraints over Joins

We must process the constraints in a **bottom-up** manner, because the columns involved in constraints $C_{\neq}$ may **overlap** with the columns in the child nodes.

Probability is not independent!

The processing strategy for each constraint in $C_{\neq}$ is the same as the constraint in $C_{=}$ (binary search & random sampling).
Data Generation

- The data generation component is responsible for **assembling tuples** based on the outputs of the **column generators**.
- The key technical **challenge** here is to meet the equality constraints over the join operators, i.e., $C^\equiv$, which involve **the dependencies among primary and foreign keys** from multiple tables.

### Compilation Step
1. Order the tables as a generation sequence
2. Decompose the query trees into constraint chains

### Assembling Step
1. Generate tuples in parallel on multiple nodes
2. Efficiently manage the join information
Compilation Step

Focus on the manipulation of primary key and foreign keys.
Assembling Step

(1) Incrementally assign a primary key;

(2) Fill values in the non-key columns by calling the random column generators;

(3) Identify the appropriate candidate referenced keys for each foreign key;

(4) Maintain the join information of the primary key.
Handling Mismatch Cases

- There are some joinability statuses of the primary key that never occur!
- Therefore, in the tuple generation, it should be avoided to search such referenced primary key.
- The main idea is to add rules to manipulate relevant FK constraints.

Join information table of rpk:

<table>
<thead>
<tr>
<th>bitmap</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFF</td>
<td>1, 5</td>
</tr>
<tr>
<td>TFF</td>
<td>6, 7</td>
</tr>
<tr>
<td>FFT</td>
<td>2, 9</td>
</tr>
<tr>
<td>TTF</td>
<td>3, 8</td>
</tr>
</tbody>
</table>

The example constraint chains of the target table:

1. Filter[...] -> FK[fk, rpk, 0.3]
2. Filter[...] -> FK[fk, rpk, 0.6] -> PK[...]
3. FK[fk, rpk, 0.2] -> Filter[... ] -> PK[...]

FK[fk, rpk, 0.17, rules:[FFT <- FT, TTF <- TF]]

An example of adjustments to FK constraints