Low-latency Spark Queries on Updatable Data

Alexandru Uta, Bogdan Ghit, Ankur Dave, Peter Boncz
Highly Dynamic Workloads on Updatable data

- Nowadays, many datasets are constantly updated
- Need to perform (near-)real-time queries on these data while being updated
- E.g.:
  - Large twitter dataset which is continuously expanding
  - Interactively analyzing this (graph-like) dataset means mostly applying joins
- Shuffling and creating hash tables for each operation is expensive
Problem: Apache Spark unfit for dynamic workloads

- Such workloads are currently run in large-scale distributed setups
- But Spark does not:
  - Store indexes
  - Support fine-grained updates/appends
  - Support fast point-lookups
  - Use such lookups for joins
Solution: Indexed DataFrame

- Indexed DataFrame supports:
  - Equality indexes
  - Fine-grained appends
  - Fast point-lookups
  - Index-based joins

<table>
<thead>
<tr>
<th>Col1</th>
<th>Col2</th>
<th>Col3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td>i</td>
</tr>
</tbody>
</table>

JOIN on Col1

<table>
<thead>
<tr>
<th>Col1</th>
<th>Col2</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>x</td>
</tr>
</tbody>
</table>
Indexed DataFrame API – extends DataFrame API

1) Index Creation:

\[
\text{indexedDF} = \text{regularDF}.createIndex(\text{columnNumber}: \text{Int})
\]

2) Append:

\[
\text{newIndexedDF} = \text{indexedDF}.appendRows(\text{regularDF}: \text{Dataframe})
\]

3) Lookups:

\[
\text{regularDF} = \text{indexedDF}.getRows(\text{key}: \text{AnyVal})
\]

4) Inner Equi-Joins:
   a) \( \text{SELECT} * \text{ from indexedDF JOIN regularDF ON indexedDF.col1 = regularDF.col2} \)
   b) \( \text{indexedDF}.join(\text{regularDF, Seq("col")}) \)
Design & Implementation (1)

- **Goal:** easy to use, easy integration with Spark
- Standalone *sbt* project, does **not** modify Spark source code
- Included in any Spark program like a standalone library
- Works with *Apache Spark & Databricks Runtime*
- Strategies and rules to convert to and support Indexed operators
- Additional rules determine whether query is indexed or not

- We extend catalyst logical and physical operators

- Indexed RowBatch RDD stores indexed data, is able to fall back to regular row RDD

- Indexed DataFrame can fall back to regular operation
Design & Implementation (3)

- Partition the data by indexed key (by range or hash)
- cTrie (concurrent trie) to store the index
- RowBatches (4MB size) to store data
- Problem: Supporting graphs => duplicate keys (when storing edges)
- Solution:
  - cTrie stores pointer to last row with same key
  - Backward pointers to previous rows
Design & Implementation (4)

- Achieving locality (i.e., not moving around indexed data):
  - **append**: partition the input data by same key then shuffle + local append
  - **join**: partition the right relation by join key then shuffle + local lookup
IndexedDF vs. Cached Spark: Various Operators

![Graph comparing IndexedDF and Spark for various operators](image-url)

- **Query types**: Join, Filter, Equality Filter, Aggregation, Projection, Scan
- **Time [ms]**
  - IndexedDF
  - Spark

The graph shows the comparison of execution time between IndexedDF and Spark for different query types.
IndexedDF vs. Cached Spark: Join, various sizes

![Graph showing time in milliseconds for different join sizes (S, M, L, XL) comparing IndexedDF and Spark. The graph indicates that IndexedDF consistently has shorter times across all join sizes.]
IndexedDF vs. Cached Spark: LDBC SNB (SF 300)

![Bar chart showing performance comparison between IndexedDF and Spark for different queries.](chart_image.png)
Conclusion and Future Work

- Presented as Demo @SIGMOD
- Indexing on Dataframes is feasible
- Good performance improvement (3X - 8.5X) for joins
- Promising performance improvement for SNB
- Explore behavior for more complex queries and different workloads
Dataset & Workloads

- Datagen scale factor 1000 (graph)
- Edge table: 1B rows
- Vertex table: 10M rows
- Workload: join edge table (indexed) with vertex table (non-indexed)

<table>
<thead>
<tr>
<th>Workload Scale</th>
<th>Edge Table</th>
<th>Vertex Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>XS</td>
<td>1B rows</td>
<td>10K rows</td>
</tr>
<tr>
<td>S</td>
<td>1B rows</td>
<td>100K rows</td>
</tr>
<tr>
<td>M</td>
<td>1B rows</td>
<td>1M rows</td>
</tr>
<tr>
<td>L</td>
<td>1B rows</td>
<td>10M rows</td>
</tr>
</tbody>
</table>
Empirical Evaluation

- **Hardware platform:**
  - DAS-5 cluster @VU Amsterdam
  - Each node 16 cores (two NUMA nodes), 64 GB RAM, FDR InfiniBand (56 Gbit/s)

- **Experiments performed:**
  - Indexed DataFrame vs. vanilla Spark (cached)
  - Multiple join sizes
  - Multiple SQL operators
  - LDBC SNB simple queries
Databricks Runtime 4.3 - 5 runs

Indexed DataFrame - 5 runs

55 seconds

2 seconds