The TileDB Array Data Storage Manager

Shiyu Huang
What is the problem?
Storage manager for multi-dimensional arrays
Why is it important?
Large scientific data naturally represented as multi-dimensional arrays

Data Sample

<table>
<thead>
<tr>
<th>ID</th>
<th>Date</th>
<th>Time</th>
<th>Latitude</th>
<th>Longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:54:50</td>
<td>37.782551</td>
<td>-122.445368</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:54:54</td>
<td>37.782745</td>
<td>-122.444586</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:54:58</td>
<td>37.782842</td>
<td>-122.443688</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:55:02</td>
<td>37.782919</td>
<td>-122.442815</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:55:06</td>
<td>37.782992</td>
<td>-122.442112</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:55:10</td>
<td>37.783101</td>
<td>-122.441461</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:55:14</td>
<td>37.783206</td>
<td>-122.440829</td>
</tr>
<tr>
<td>1</td>
<td>3/7/07</td>
<td>10:55:18</td>
<td>37.783273</td>
<td>-122.440324</td>
</tr>
</tbody>
</table>

Overall
1.1M distinct readings
25,000 distinct trips.
Why is it hard?
Read & Write on large array

- Array representation
- Compression
- Parallel access
- Performance
Why existing solutions do not work?
HDF5

- Hard to identify and manage dense array
- In-place write

Array-oriented DB

- Regular dimensional chunk as atomic unit

Relational databases

- Encoding the element indices as extra table columns
Core intuition for the solution?
Data model

- Dimensions
- Coordinates
- Cells
- Attributes
Global cell order

Co-located data according to the characteristics of the data
Data tiles

Dense arrays
- Space tile
- equi-sized hyper-rectangles

Sparse arrays
- Capacity
- Minimum bounding rectangle
- Each non-empty cell in one data tile
Fragments

- Timestamped snapshot of a batch of updates
Physical Organization

Array -> Directory

Fragment -> Sub-directory

Fixed-sized attribute 1

Values of Variable sized attribute 1

Array schema

Starting offsets of variable-sized attribute 1
Bookkeeping metadata:
Minimum-bound rectangle
Bounding coordinates
Read operation

- Dense array: visit each space tile in global order
- Sparse array: visit each range that start before the minimum end bounding coordinate
Multiple fragments?

Algorithm!
Sort disjoint ranges on global cell order

- <start coordinate, end coordinate, fragment id>
- Each range appears contiguously on the disk

```
<table>
<thead>
<tr>
<th>Fragment #1 (dense)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>bb</td>
</tr>
<tr>
<td></td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>ff</td>
</tr>
<tr>
<td>2</td>
<td>ccc</td>
</tr>
<tr>
<td></td>
<td>dddd</td>
</tr>
<tr>
<td></td>
<td>ggg</td>
</tr>
<tr>
<td></td>
<td>hhhh</td>
</tr>
<tr>
<td>3</td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>jj</td>
</tr>
<tr>
<td></td>
<td>m</td>
</tr>
<tr>
<td></td>
<td>nn</td>
</tr>
<tr>
<td>4</td>
<td>ikkk</td>
</tr>
<tr>
<td></td>
<td>llll</td>
</tr>
<tr>
<td></td>
<td>ooo</td>
</tr>
<tr>
<td></td>
<td>pppp</td>
</tr>
</tbody>
</table>
```

e.g. querying <2,5>:
Get range <2,3>, <4,5>
Priority queue

Compare on SC
When tied, give higher fid priority
Operations in the priority queue

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>popped</td>
<td>top</td>
</tr>
<tr>
<td></td>
<td>insert to result</td>
<td></td>
</tr>
<tr>
<td>global cell order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii)</td>
<td>popped [ ]</td>
<td>top</td>
</tr>
<tr>
<td></td>
<td>insert to result</td>
<td>re-insert to pq</td>
</tr>
<tr>
<td>global cell order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii)</td>
<td>check against new top</td>
<td></td>
</tr>
<tr>
<td>popped</td>
<td>top</td>
<td></td>
</tr>
<tr>
<td></td>
<td>discard</td>
<td>re-insert to pq</td>
</tr>
<tr>
<td>global cell order</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iv)</td>
<td>popped [x]</td>
<td>top</td>
</tr>
<tr>
<td></td>
<td>re-insert to pq as dense</td>
<td>re-insert to pq as sparse</td>
</tr>
<tr>
<td>global cell order</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Priority queue

Compare on SC

When tied, give higher fid priority
Write - dense fragment

- The user:
  - populates one buffer per attribute, storing the cell values respecting the global cell order

- Write function:
  - Append the values from the buffers into the corresponding attribute file
Sparse fragment

- **Mode:**
  - User provides sorted buffer
  - User provides unsorted buffer, TileDB sorts it internally

- **Buffer:**
  - Only non-empty cells
  - Extra buffer with coordinates
  - Extra write state information

- **Deletion:**
  - Insertions of empty cells
Consolidation

Read operation

Delete the old fragments

Write the retrieved cells to a new fragment
Parallel Programming

- Concurrent write: Each process/thread creates a separate fragment, no locking necessary
- Concurrent read:
  - Multiple process: separate bookkeeping data and state, no locking
  - Multiple thread: One bookkeeping data, only lock on it
- Mixed read and write: Special file indicates if the fragment is visible
- Background consolidation:
  - Get the lock when delete the old fragment, release the lock after new fragment is visible
Does the paper prove its claims?
Does the paper prove its claims?

- Clear description of the physical layout and functions
- Logical justification of the design decision
- Comprehensive evaluations
Analysis/experiments?
Dense Array

One Core

Parallel

LOAD

UPDATE

(a) vs. dataset size (HDD)

(b) vs. # instances (SSD)

(a) vs. # updates (HDD)

(b) vs. # instances (SSD)
Read subarray

(a) vs. subarray type (HDD)

(b) vs. # tiles (HDD)

(c) vs. # elements (HDD)

(d) vs. # instances (SSD)
Fragments and consolidation

(a) Subarray time (HDD)  (b) Consolidation time (HDD)

After consolidation
Sparse array

- Load

(a) vs. dataset size (HDD)  
(b) vs. # instances (SSD)

One core  Parallel
Subarray read

(a) DQ vs. result size (HDD)  (b) SQ vs # result size (HDD)

(c) DQ vs. # instances (SSD)  (d) SQ vs. # instances (SSD)
Gaps in the logic/proof?
Possible next step
Distributed version of database?
Versioning?