ArrayStore

A STORAGE MANAGER FOR
COMPLEX PARALLEL ARRAY PROCESSING
MARTIN HIRSCHBERGER 19.11.2018
ArrayStore – Multidimensional Storage system

What is ArrayStore?
◦ Data management system for Multidimensional arrays
◦ Supports parallel processing of data
◦ Performs array-specific operations such as feature extraction, smoothing, clustering

Why ArrayStore?
◦ Inefficiency of simulating multidimensional arrays on top of the relational model
◦ Support of growing data management needs

Using examples:
◦ 3D astronomy
◦ 6D flow-cytometer datasets
Array chunking

- extract a subset of an array
e.g. array slicing, dicing

- binary array operations
  e.g. joins, cross-match

- access data from adjacent partitions
  e.g. Canopy Clustering
Canopy Clustering

Algorithm:

1. Pick and remove Random point from set of coordinates
2. Create a Canopy containing this point
3. Iterate through the remaining points of the set.
   1. Distance between center point and current point < T1 ➔ add point to Canopy
   2. Distance is < T2 < T1 ➔ remove the point from the set.
4. Redo 1. with remaining points till set is empty

Used as preclustering for more expensive clustering methods (e.g. K-Means Clustering)

➔ Reducing the number of more expensive distance measurement
Parallel Clustering and Overlap needs

Processing the Canopy-Algorithm parallel on each chunk

⇒ Points on the border missing in the Cluster

Strategies needed to added the missing points to the cluster:

◦ Ignoring overlap need and post-process cluster
◦ Provide overlap data
Strategy: No Overlap

Processing each chunk alone ignoring overlap needs

⇒ expensive postprocess necessary
Strategy: Single-Layer

Extract overlap area from neighboring chunks

➔ No post-processing phase

Canopy only needs Overlap of T1

But:

◦ small overlap can impose huge overhead
◦ E.g. 10% larger along each dimension (only 5% on each side)

➔ total I/O and CPU overhead 33% for a 3D chunk, over 75% for a 6D chunk
Strategy: Multi-Layer using two-level storage

Collecting overlap data via Two-level Storage access

➔ only Chunks covering the overlap region are loaded

No overlap region needs to be configured ahead of time

Inefficiencies of Multi-Layer:

◦ To requests overlap data within a neighbouring chunk the entire chunk must be read
◦ overlap layers processed at the granularity of tiles

➔ Using Materialized Overlap-Views

Algorithm 1 Multi-Layer Overlap over Two-level Storage

1: Multi-Layer Overlap over Two-level Storage
2: Input: chunk core_chunk and predicate overlap_region.
3: Output: chunk result_chunk containing all overlap_region.
4: ochunkSet ← all chunks overlapping overlap_region.
5: tileSet ← ∅
6: for all Chunk ochunk_i in ochunkSet - core_chunk do
7:    Load ochunk_i into memory.
8:    tis ← all tiles in ochunk_i overlapping overlap_region.
9:    tileSet ← tileSet ∪ tis
10: end for
11: Combine tilesSet into one chunk result_chunk.
12: return result_chunk.
Strategy: Overlap-Views

- Small Layers in form of onion-skin around the chunk
- Only Layers covering requested area are passed to operator
- Need predefined Overlap-Views for each chunk

Algorithm 2 Multi-Layer Overlap using Overlap Views

1: Multi-Layer Overlap using Overlap Views
2: Input: chunk core_chunk and predicate overlap_region.
3: Output: chunk result_chunk containing requested overlap data.
4: Identify materialized view $M$ to use.
5: $L \leftarrow$ layers $l_i \in M$ that overlap overlap_region.
6: Initialize an empty result_chunk
7: for all Layer $l_i \in L$ do
8:  Load layer $l_i$ into memory.
9:  Add $l_i$ to result_chunk.
10: end for
11: return result_chunk.
Benchmarks Chunking

Subsample IREG and REG chunks, 1% of the array volume, worst case shape for REG

<table>
<thead>
<tr>
<th>Type</th>
<th>I/O time (Sec)</th>
<th>Proc. time (Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(REG,4096)</td>
<td>28</td>
<td>115</td>
</tr>
<tr>
<td>(REG,262144)</td>
<td>46</td>
<td>51</td>
</tr>
<tr>
<td>(REG,2097152)</td>
<td>90</td>
<td>66</td>
</tr>
<tr>
<td>(REG-REG,4096-2097152)</td>
<td>28</td>
<td>64</td>
</tr>
</tbody>
</table>
Benchmarks Canopy Clustering

<table>
<thead>
<tr>
<th></th>
<th>Time in s</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Parallel</td>
<td></td>
</tr>
<tr>
<td>No Overlap</td>
<td>2</td>
</tr>
<tr>
<td>Single Layer</td>
<td>1.5</td>
</tr>
<tr>
<td>Multi Layer</td>
<td>0.5</td>
</tr>
<tr>
<td>Overlap View</td>
<td></td>
</tr>
</tbody>
</table>
References
