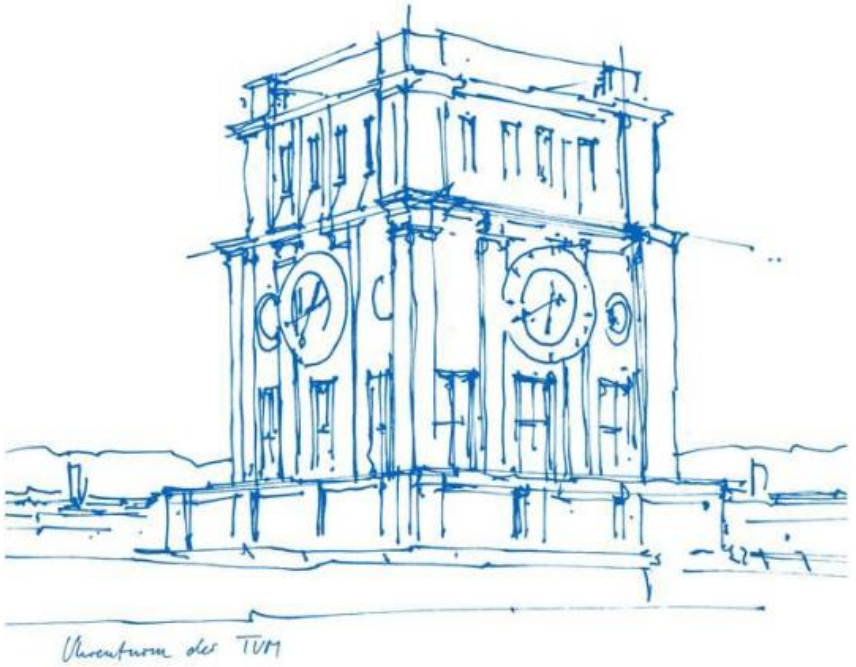
Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München



ArrayStore

A STORAGE MANAGER FOR COMPLEX PARALLEL ARRAY PROCESSING

MARTIN HIRSCHBERGER 19.11.2018

Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München

ArrayStore – Multidimensional Storage system

What is ArrayStore?

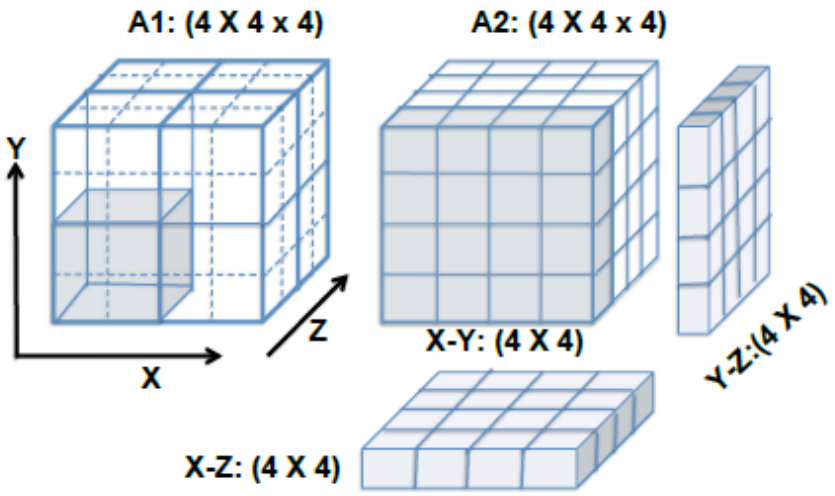
- Data management system for Multidimensionalarrays
- Supports parallel processing of data
- Performs array-specfic operations such as feature extraction, smoothing, clustering

Why ArrayStore?

- Inefficiency of simulating multidimensional arrays on top of the relational model
- Support of growing data management needs

Using examples:

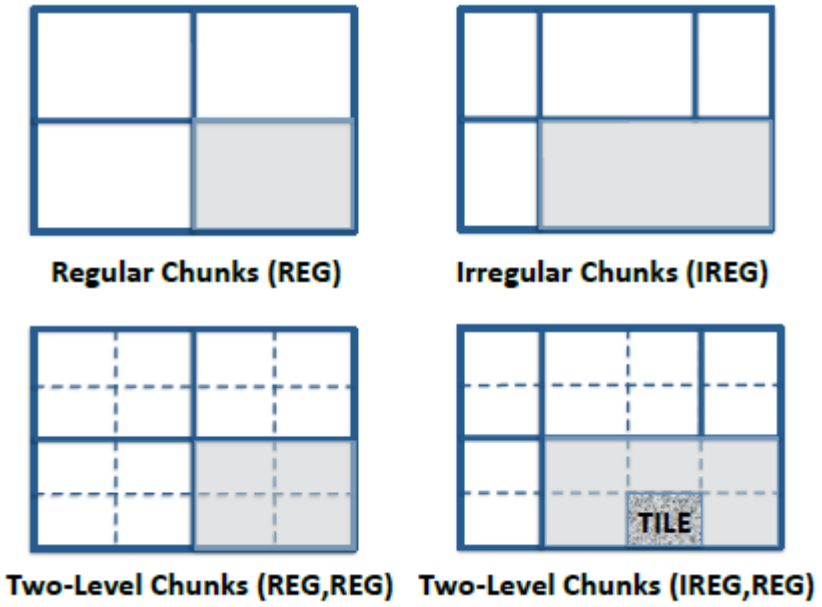
- 3D astronomy
- 6D flow-cytometer datasets



Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München

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**



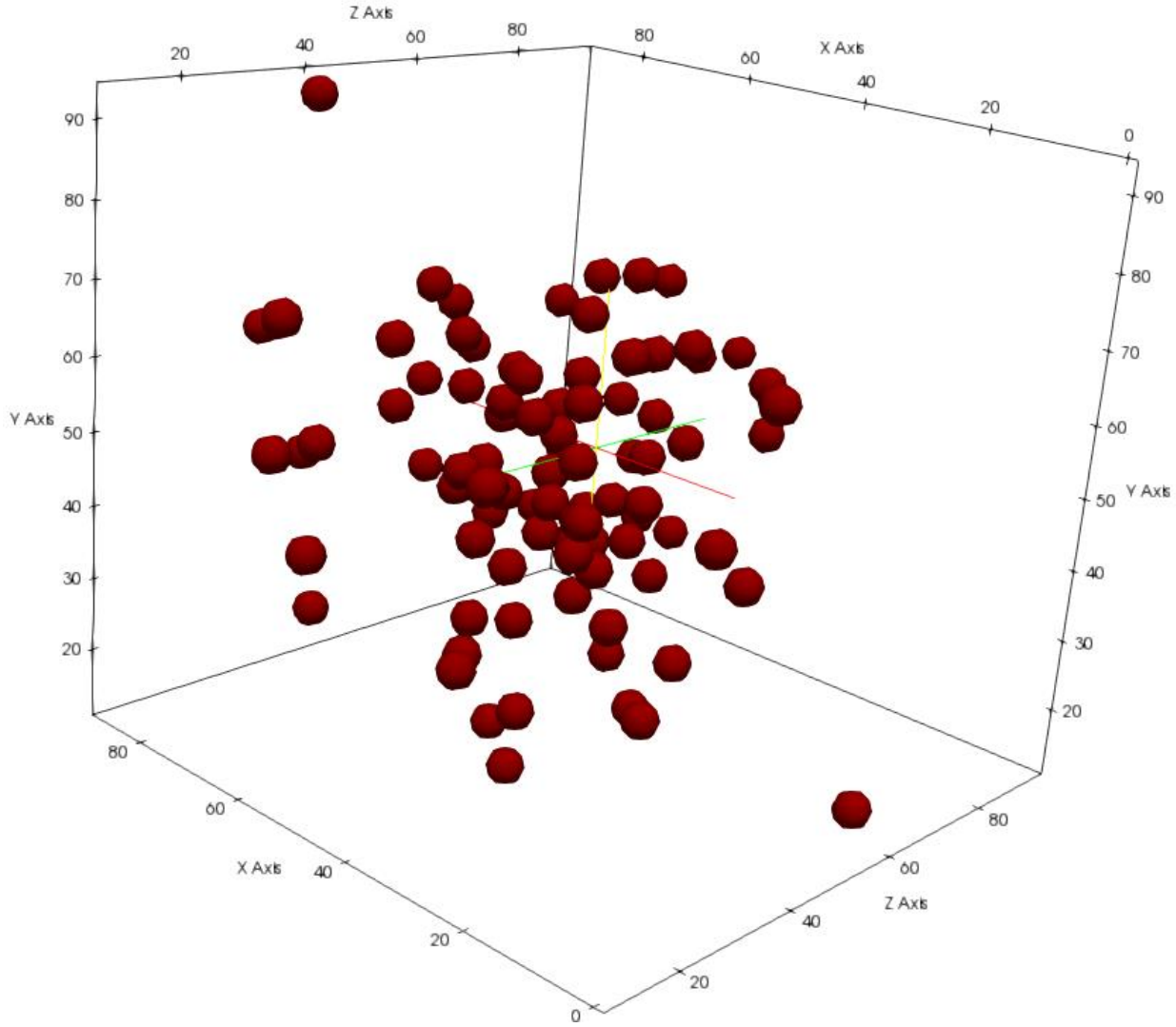
Canopy Clustering

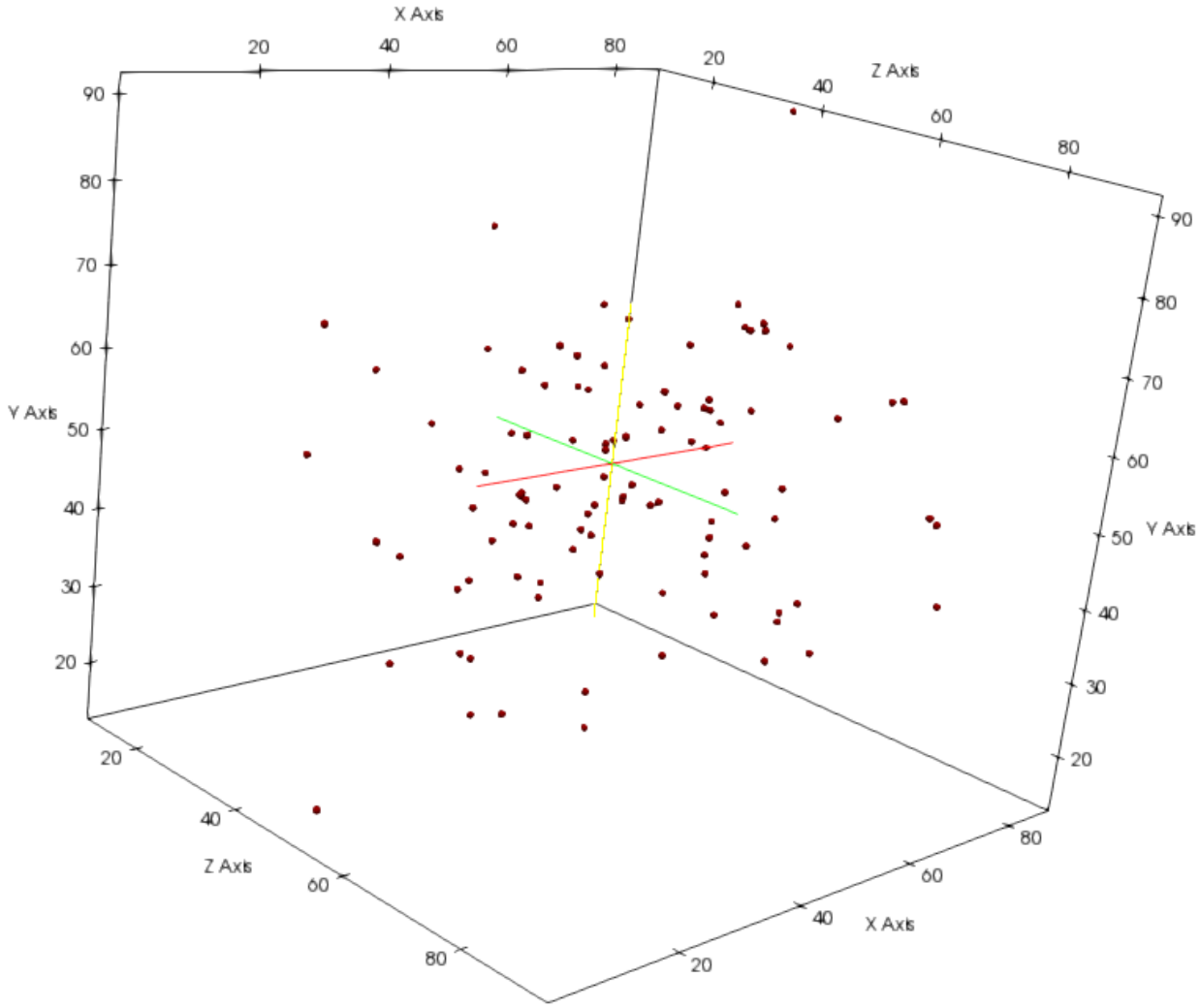
Algorithm:

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


Z Axis
X Axis
Y Axis
Y Axis
X Axis
Z Axis


X Axis
Z Axis
Y Axis
Y Axis
Z Axis
X Axis

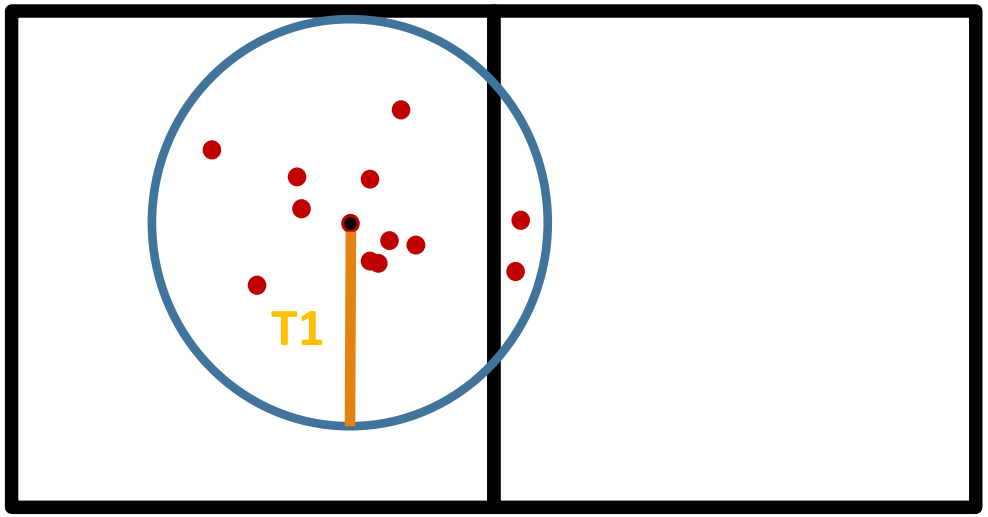
Parallel Clustering and Overlap needs

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:

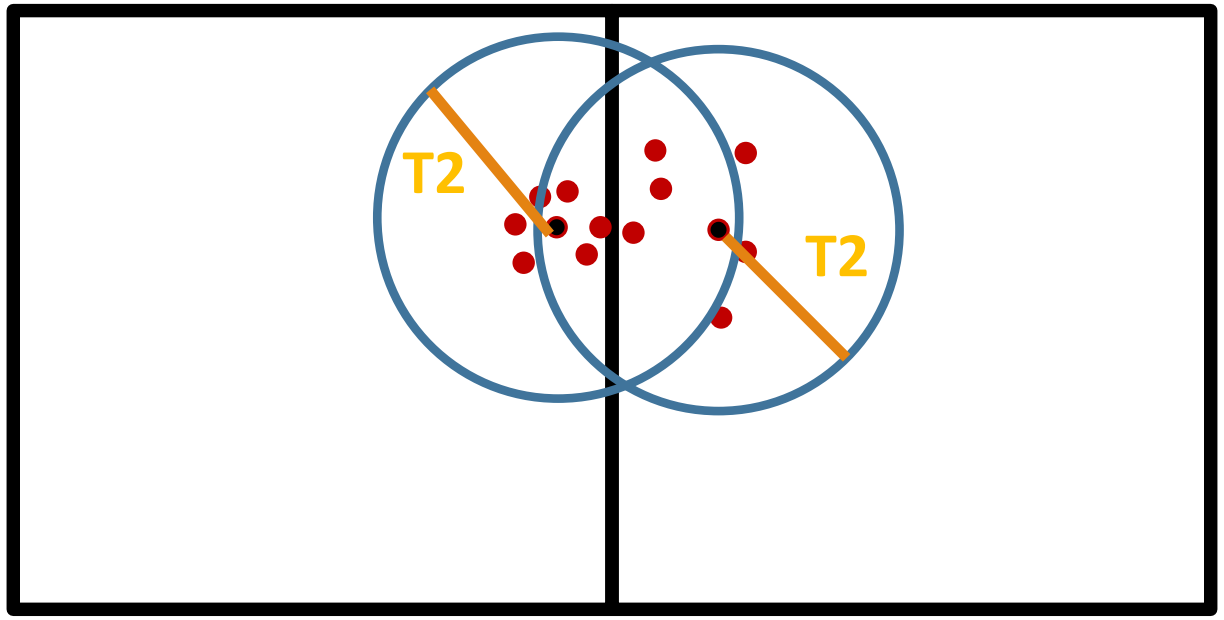
- Ignoring overlap need and post-process cluster
- Provide overlap data



Strategy: No Overlap

Processing each chunk alone ignoring overlap needs

➔expensive postprocess necessary



Strategy: Single-Layer

Extract overlap area from neighboring chunks

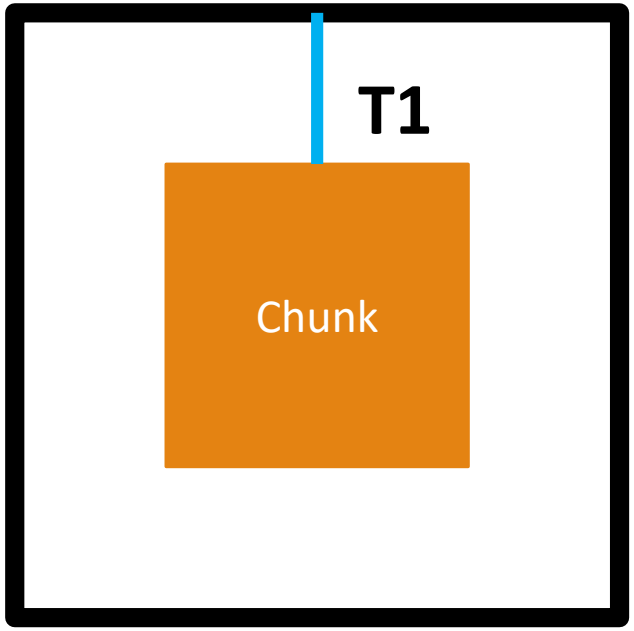
➔No post-processing phase

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 tiles.
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.
```

[1]

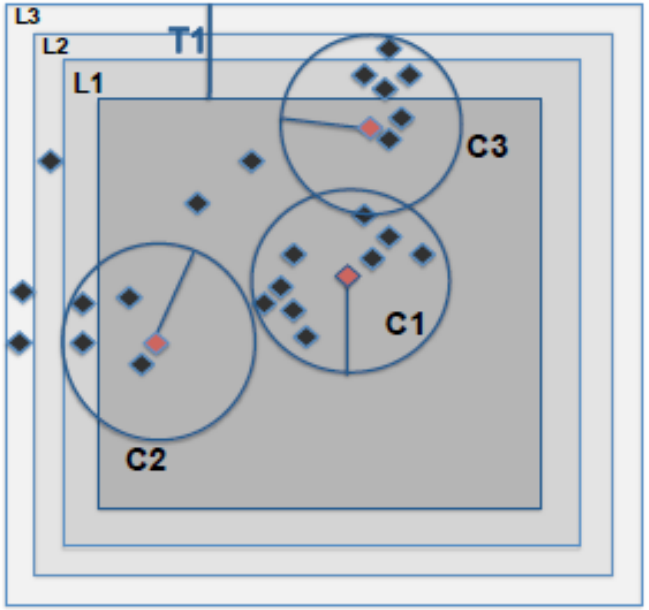
Strategy: Overlap-Views

- 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 ← layers l_i ∈ M that overlap overlap_region.
6: Initialize an empty result_chunk
7: for all Layer l_i ∈ L do
8:     Load layer l_i into memory.
9:     Add l_i to result_chunk.
10: end for
11: return result_chunk.
```

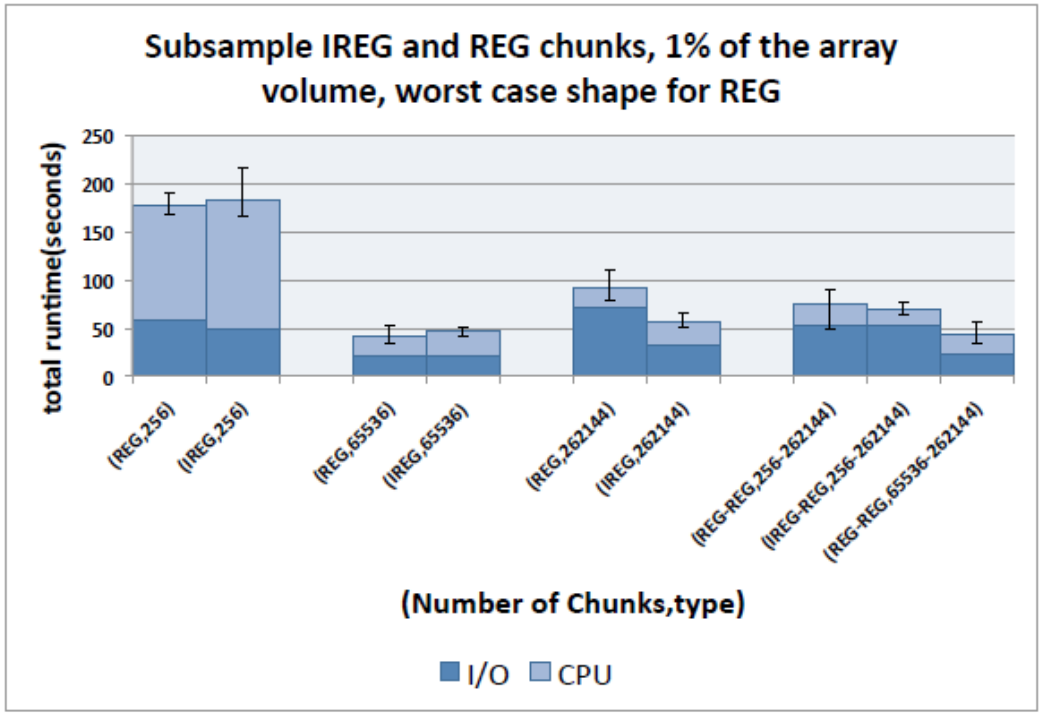
[1]



[1]

Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München

Benchmarks Chunking



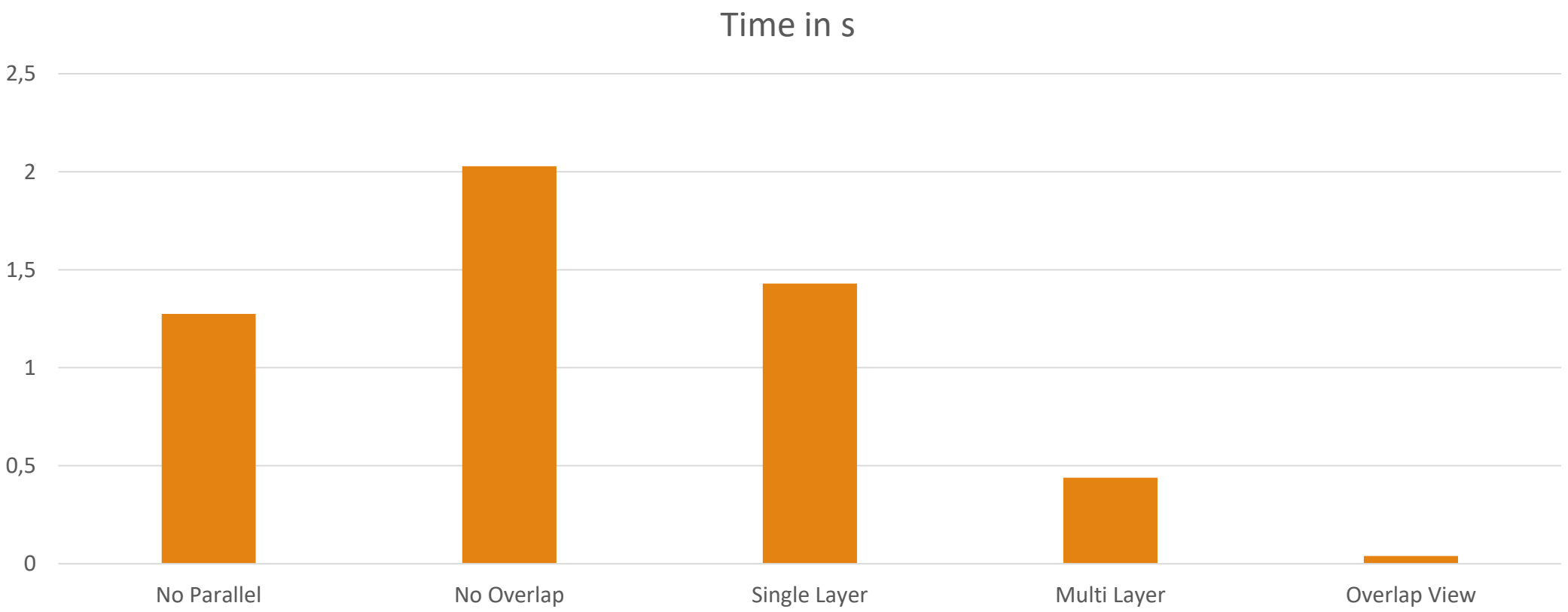
[1]

Type	I/O time (Sec)	Proc. time (Sec)
(REG,4096)	28	115
(REG,262144)	46	51
(REG,2097152)	90	66
(REG-REG,4096-2097152)	28	64

[1]

Lehrstuhl für Datenbanksysteme
Fakultät für Informatik
Technische Universität München

Benchmarks Canopy Clustering



References

[1] Soroush et al.: ArrayStore: A Storage Manager for Complex Parallel Array Processing , In: SIGMOD 2011

[2] http://mahout.apache.org