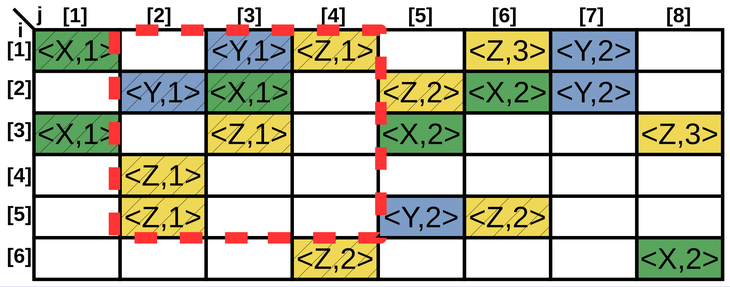
Distributed Caching for Processing Raw Arrays

Weijie Zhao[1], Florin Rusu[1,2], Bin Dong[2], Kesheng Wu[2], Anna Ho[3], and Peter Nugent[2]

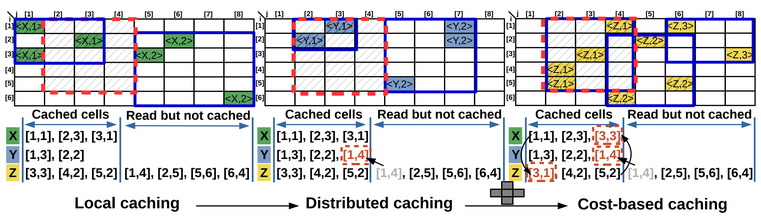
[1]University of California Merced, [2]Lawrence Berkeley National Laboratory, [3]CalTech

wzhao23,frusu@ucmerced.edu; dbin,kwu@lbl.gov; ah@astro.caltech.edu; penugent@lbl.gov

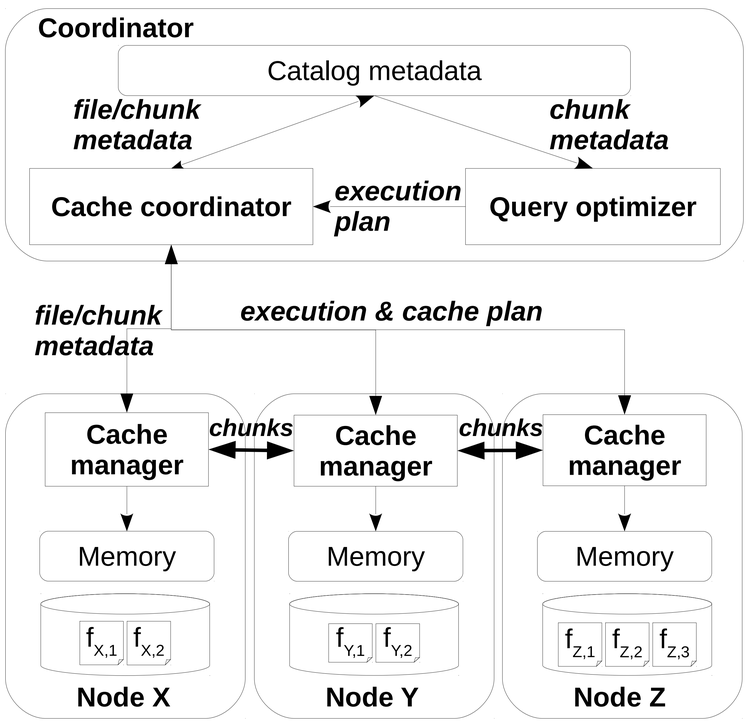
Raw Array Data Model



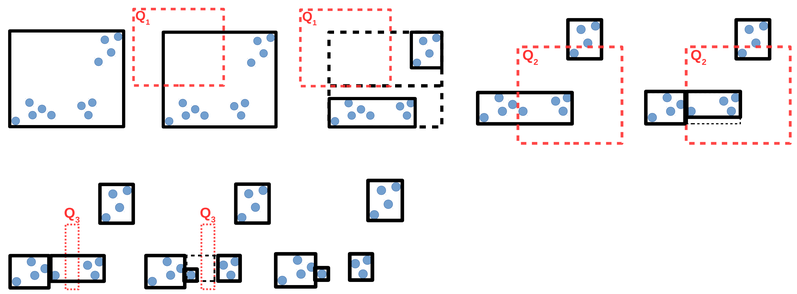
Raw Array Distributed Caching



Distributed Caching Architecture



Raw Array Chunking



When to split?

- Chunk is sufficiently large.
- Query subarray does not contain any cells.

How to split?

- According to the query subarray boundaries?

Algorithm 1 Chunk Split

Input: Chunk α with bounding box $\mathcal{BB}_\alpha$ that intersects query subarray $\boldsymbol{Q}$; Minimum number of cells threshold *MinC*

Output: Chunks β and γ after splitting α

1: **if**(cells in $\alpha <$ *MinC*) **and** ($\exists$ cell in $\alpha \in \boldsymbol{Q}$) **then return**
2: min_vol = $+\infty$
3: **for each** boundary $\boldsymbol{b} \in \boldsymbol{Q}$ that intersects with $\mathcal{BB}_\alpha$ **do**
4: $(\beta_{\boldsymbol{b}}, \gamma_{\boldsymbol{b}}) \leftarrow$ split cells in α into two sets by boundary $\boldsymbol{b}$
5: **if** vol($\beta_{\boldsymbol{b}}$) + vol($\gamma_{\boldsymbol{b}}$) $<$ min_vol **then**
6: min_vol $\leftarrow$ vol($\beta_{\boldsymbol{b}}$) + vol($\gamma_{\boldsymbol{b}}$)
7: $\beta \leftarrow$ bounding_box($\beta_{\boldsymbol{b}}$), $\gamma \leftarrow$ bounding_box($\gamma_{\boldsymbol{b}}$)
8: **end if**
9: **end for**

Cost-Based Caching

Cache Eviction

We must scan a file entirely even if only one accessed chunk isn't cached.
→ We aim to cache all the queried chunks in a file.

$$cost_{evict}(\boldsymbol{Q_l}, \boldsymbol{f_i}, \{\boldsymbol{C_j}\}) = \boldsymbol{w_{Q_l}} \cdot \frac{size(\boldsymbol{f_i})}{\sum size(uncached\ \boldsymbol{C_j})}$$

Cache Placement

We piggyback on the replication induced by query execution.

$$cost_{placement}(\boldsymbol{C_i}, \boldsymbol{n}, \boldsymbol{P'}, \boldsymbol{W}) = \sum_{\boldsymbol{Q} \in \boldsymbol{W}} \boldsymbol{w_Q} \cdot \left| \boldsymbol{C_j} \in \boldsymbol{P'_n} \wedge (\boldsymbol{C_i}, \boldsymbol{C_j}) \in \boldsymbol{Q} \right|$$

$\boldsymbol{C_i}$: current chunk to place, $\boldsymbol{n}$: candidate node – typically chosen from where a replica of $\boldsymbol{C_i}$ exists
$\boldsymbol{P'}$: location of already placed chunks, $\boldsymbol{W}$: query workload.

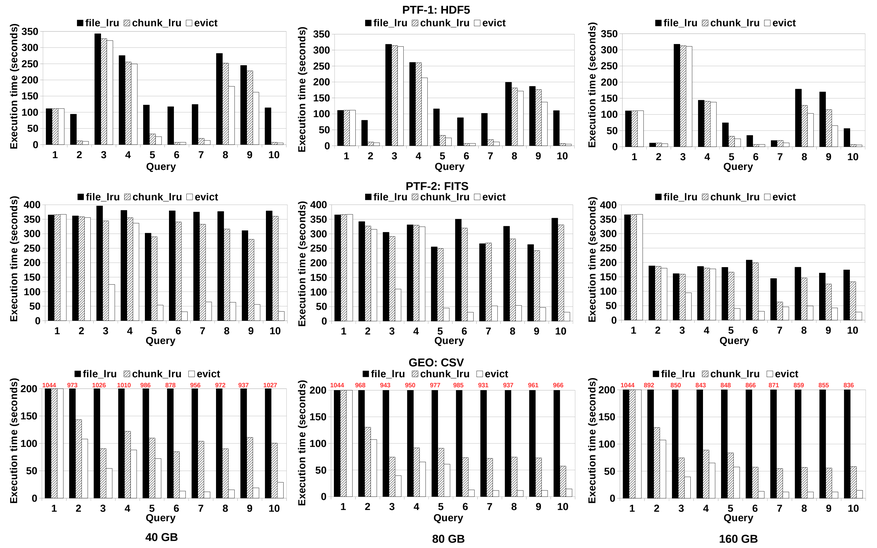
Experiments

PTF catalog: `PTF[time=1,153064;ra=1,100000;dec=1,50000]`
1 billion objects, 343 GB in CSV, 262 GB in HDF5, and 221 GB in FITS

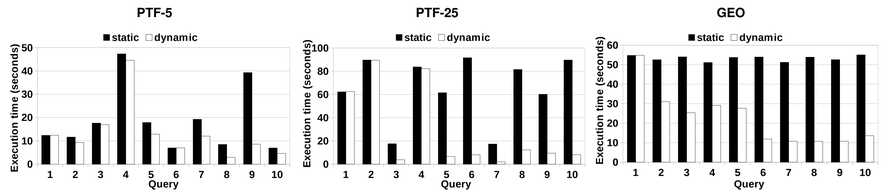
LinkedGeoData: `GEO[long=1,100000;lat=1,50000]`
30 billion objects, 1.6 TB in CSV.

3 query patterns: real workload, shifting ranges and alternative queries.

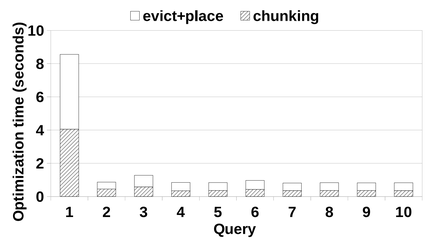
Query execution time:



Similarity join execution time:



Optimization time



`evict` v.s `chunk_lru` improvement