Administrivia

- Project
  - time to find a partner and start looking for an application to implement with DataCutter
    - code can be partitioned into interacting components (a pipeline)
    - also need input data (< 500MB)
  - DataCutter installed on redleader in ~chansen/work/DataCutter-2.1/
    - Documentation is in doc/, binaries in bin/, etc.
- Midterm date announced soon

Declustering

Moon et. al. – Scalability Analysis of Declustering Methods for Multidimensional Range Queries

Declustering

- Distributing files across multiple disks
  - to exploit parallelism in disk accesses
  - to minimize query response time, maximize system throughput
- Used in relational DB systems, RAIDs, parallel file systems, …

Cartesian Product Files

- Multi-attribute file structure effective for partial- and best-match queries
- Declustering methods include:
  - Disk Modulo (DM)
  - Fieldwise Xor (FX)
  - Error Correcting Codes (ECC)
  - Hilbert Curve-Allocation Method (HCAM)
  - Vector-based declustering method
  - Graph partitioning-based algorithms

Range Queries

- Query provides a range of values for one or more attributes
  - like Titan or Virtual Microscope queries
- This paper studies the scalability of multidimensional declustering methods, as the number of disks increases
  - but we’re mostly interested in the algorithms, not the scalability analysis
Cartesian product files

- Pretty much what we talked about for the sensor and simulation application data
- Each record is a tuple consisting of a set of fields/attributes
- The records are partitioned into buckets/blocks/chunks of records according to the values of the attributes
  - subspaces of the attribute space are stored in different buckets

Disk Modulo (DM)

- Assign bucket \([i_1, i_2, \ldots, i_d]\) to disk number \((i_1 + i_2 + \ldots + i_d) \mod M\), where \(M\) is number of disks
- Optimal for many cases of partial match queries (those that specify matching some, but not all, of the attributes to particular values)

Fieldwise XOR (FX)

- Replace + with bitwise exclusive-or on bucket coordinates
- Assign bucket \([i_1, i_2, \ldots, i_d]\) to disk number \((i_1 \oplus i_2 \oplus \ldots \oplus i_d)\)
- Optimal when number of disks \(M\), and size of each attribute \(i\), are powers of 2

Hilbert Curve Allocation Method (HCAM)

- Use space filling curves
  - visits all points in a \(d\)-dimensional space exactly once, without crossing itself
  - used to linearize a set of buckets, then assign buckets to disks round-robin
- Assign bucket \([i_1, i_2, \ldots, i_d]\) to disk number \(H(i_1, i_2, \ldots, i_d) \mod M\), where \(H\) maps bucket coordinates to a Hilbert linear ordering
- Performs well for small range queries and large \(M\), shown empirically

Vector-based Method

- Generate a pair of vectors (for a 2D domain) \(u = (a, b)\) and \(v = (c, d)\) – how?
  - Only works for 2 dimensional Cartesian product files
- Assign all buckets with coordinates of form \([x+m*a+n*c, y+m*b+n*d]\) for any integers \(m\) and \(n\) to same disk as subspace \([x, y]\)

Proximity-based algorithm

- Turn the declustering problem into a graph partitioning problem
  - One vertex per subspace, one edge for every pair of subspaces
  - Each edge weight assigned based on probability that the vertices it connects will both be accessed by a query
  - Declustering for \(M\) disks is an \(M\)-way partition of the graph
  - Variant of the Max-Cut problem, which is NP-complete
Minimax spanning tree algorithm

- O(N^2) disk accesses to decluster file with N buckets
  - Previous algorithms are O(N)
  - Other heuristics are even worse
- Generates perfectly balanced partitions
  - Each disk gets \( \lfloor N/M \rfloor \) buckets
- If two buckets likely to be accessed together, then very unlikely they will be assigned to same disk

Minimax algorithm (cont.)

- Prim’s algorithm - idea is to expand existing minimal spanning tree by incrementally selecting min-cost edge between vertices already in the tree and those not yet in the tree
  - Doesn’t ensure that aggregate cost (sum of all edge weights) due to new vertex is minimized
- Minimax instead uses min of max cost
  - For all unselected vertices, compute max of all edge weights between it and vertices already selected
  - Pick the vertex with the least value to add
- To generate partitions, grow M spanning trees, selecting vertices for them in round-robin order
  - Randomly select M vertices to get started

Minimax algorithm (cont.)

- Algorithm doesn’t guarantee that two buckets close to each other are assigned to different disks (trees)
  - But experimental results show it rarely happens
- Edge weights generated with proximity index - works well for spatial objects compared to Euclidean distance, which is suitable for point objects

Conclusions

- Graph-based minimax algorithm can generate better declustering than other methods
  - Takes into account spatial proximity of different buckets/clunks
- But graph algorithm is expensive
  - O(N^2) hurts when N is very large
  - And doesn’t generate that much better declustering than O(N) methods (e.g., HCAM, which performs very well in practice)
  - Which is why we’ve been using HCAM

Clustering

- Idea is to order blocks on disk to make it more likely to read consecutive blocks
  - Place blocks that are close in a multi-dimensional space near each other in the 1D disk space/file
  - For performance reasons, to reduce disk seeks
  - Take advantage of locality in access patterns to the multi-dim data (range queries)

Clustering

Moon et. al., Hilbert Space Filling Curves
**Mapping functions**

**Space filling curves**

- **z-ordering (z-curve)**
  - Interleave bits from the multi-dimensional coordinates

- **Gray-coded curve**
  - Use Gray coding on the interleaved bits from the z-ordering

- **Hilbert curve**
  - Use Hilbert numbers
  - Works best to preserve locality

---

**Cluster**

- A group of points (blocks) inside a query that are consecutively connected by a space-filling curve
  - If each grid point corresponds to a disk block, each cluster needed to respond to a query requires a disk seek

---

**The problem for the paper**

- Given a d-dimensional range query, find the average number of clusters inside the polyhedron for the Hilbert curve
  - Same question as how many disk seeks/reads are required to answer the query

- Read the paper for a (complex) answer
  - Bottom line is that, from experiments, it does better than z- or Gray-curves, both on average and worst case – on average a lot better for the queries shown