Lazy Analytics: Let Other Queries Do the Work for You

William Jannen, Michael Bender, Martin Farach-Colton, Rob Johnson, Bradley C. Kuszmaul, Donald E. Porter

Stony Brook University, Rutgers University, Massachusetts Institute of Technology
Two Common Types of Queries

- Small queries that must be answered quickly
  - High priority, latency sensitive tasks
  - Fetching data for page loads

- Large analytic queries
  - Might take several hours in the best case
  - Can be delayed without harming their value to the business
  - Scanning customer databases to identify fraud patterns
• Problem: Queries Compete for I/O

• Large queries delay latency-sensitive tasks
  - Does not make sense to run both types of queries on the same machine
• Independent large queries do not benefit from shared working
Ideal System

➢ Independently schedule sub-parts of large operations
➢ Piggy-back I/O on other tasks
Ideal System

➢ Independently schedule sub-parts of large operations
➢ Piggy-back I/O on other tasks
➢ Schedule related tasks together
Ideal System

- Independently schedule sub-parts of large operations
- Piggy-back I/O on other tasks
- Schedule related tasks together
Ideal System

- Flexibility to schedule sub-parts of large tasks opportunistically
- Maximizes benefits of caching
  - Large tasks should piggy back on I/O of small tasks
  - Tasks should share working sets when tasks overlap
- Use MVCC to provide transactional semantics

Insight: write-optimized dictionaries already implement this functionality for writes.
Derange Queries

Give to queries the I/O savings that write optimization gives inserts

- Piggyback I/O on other operations
- Can execute lazily
  - System has flexibility to defer tasks until convenient or required
  - Can schedule parts of queries independently
  - Still operate on a snapshot of the data consistent with query time
In Rest of This Talk

- The derange query model with an example
- How to encode queries as “inserts” in a write-optimized dictionary
- Some asymptotic performance analysis (DAM Model)
- Particularly beneficial use cases
derange(R, Filter, Map, Fold, k)

- **R** - the input range
- **Filter** - predicate to remove records that do not meet a criteria
- **Map** - function to apply to each record
- **Fold** – commutative, associative function to propagate results
- **k** – (key, value) pair where results are accumulated

Derange queries map a function over a range of records and lazily aggregate the results.
Example: Online Marketplace

Inventory Database

Item {
  productId : num
  warehouse : address
  quantity : num
  value : num
  price : num
}


Example: Online Marketplace

derange(R, Filter, Map, Fold, k)

- **R** = \((-∞, ∞)\)
- **Filter** = \{ return Item.warehouse != NY \}
- **Map** = \{ return Item.quantity * Item.value \}
- **Fold** = \{ totalValue += result \}
- **k** = “InventoryAt||TIMESTAMP”

What is the total value of all products stored in NY warehouses?
In Rest of This Talk

- The derange query model with an example
- How to encode queries as “inserts” in a write-optimized dictionary
- Some asymptotic performance analysis (DAM Model)
- Particularly beneficial use cases
Write-optimized Dictionaries

- High performance indexes by aggregating updates
  - Lookup performance is comparable to traditional data structures
  - Inserts are orders of magnitude faster
- Used by some of the fastest databases\(^1\) and file systems\(^2\) to speed up writes
- \(B^\epsilon\)-Tree is an ideal candidate for implementing derange queries

\(^1\) LevelDB, HBase, Cassandra, TokuDB, TableFS, KVFS, TokuFS, BetrFS
$B^\varepsilon$-Trees Are a Better Search Tree

leaves

children

leaves
All data is inserted to the root node’s buffer.
Bε-Trees

When a buffer fills, contents are flushed to children.
Bε-Trees

Flushes can cascade if not enough room in child nodes
B^ε-Trees

Height in the tree preserves the order of updates
Bε-Trees

derange(R, Filter, Map, Fold, k)

- \( R = (-\infty, \infty) \)
- \( \text{Filter} = \text{return Item.\text{warehouse} == NY} \)
- \( \text{Map} = \text{return Item.\text{quantity} * Item.\text{value}} \)
- \( \text{Fold} = \text{totalValue += result} \)
- \( k = \text{“InventoryAt||TIMESTAMP”} \)
$\mathbb{B}^{\varepsilon}$-Trees

During a flush, the message is split into subranges.
B^ε-Trees

Each subrange moves down the tree independently.
$B^\varepsilon$-Trees
The derange query finally executes when it reaches a leaf node.
B^ε-tree + Derange Query Recap

- Inserts are buffered in the root and flushed from node to node
  - Many application-level updates are aggregated into each I/O
- We can encode a derange query as an “insert” message
  - Treated like any other message within the tree
  - Evaluated when they reach a leaf node
- Derange queries split and travel down the tree independently
- Results are lazily folded into the final result
In Rest of This Talk

- The derange query model with an example
- How to encode queries as “inserts” in a write-optimized dictionary
- Some asymptotic performance analysis (DAM Model)
- Particularly beneficial use cases
Performance

Point Query:

Range Query:

Derange Query: ???
Asymptotic Analysis Recap

- The **batching factor** $\left( B^{1-\epsilon} \right)$ *divides* the insert cost
- By encoding queries as inserts, we bring these gains to queries
- Analysis is specific (query is allowed to take arbitrarily long)
  - Plan to generalize
In Rest of This Talk

- The derange query model with an example
- How to encode queries as “inserts” in a write-optimized dictionary
- Some asymptotic performance analysis (DAM Model)
- Particularly beneficial use cases
Opportunity: Overlapping Ranges

- Queries with overlapping ranges travel down the tree together

Derange queries are a natural way to schedule tasks by to working set.

- Partial Overlap
- Complete Overlap

- Beneficial scheduling is transparent to application
- Removes complexity of query planning
Opportunity: Fine Granularity Reporting

- Efficient point-in-time computations
  - Even if work is deferred, computations are done on the view of the data at the time that the query was issued
- If data hasn’t changed, 1 I/O satisfies all queries

Derange queries can increase the granularity of reporting at low cost.
Takeaways

- We can use write-optimization to reduce the cost of queries
- Low-cost analytics without harming latency-sensitive operations
- Asymptotic analysis for some cases (more work to be done)
- Exciting opportunities for scheduling and workload management