1-Bucket-Theta: Map

- Input: tuple \( x \in S \cup T \), matrix-to-reducer mapping lookup table
- 1. If \( x \in S \) then
  - matrixRow = random\( \{1, |S|\} \)
  - For all regionID in lookup.getRegions(matrixRow)
    - Output (regionID, \( \{x, "S"\} \))
- 2. Else
  - matrixCol = random\( \{1, |T|\} \)
  - For all regionID in lookup.getRegions(matrixCol)
    - Output (regionID, \( \{x, "T"\} \))

1-Bucket-Theta: Reduce

- Input: \((ID, [(x_i, \text{origin}_i), ..., (x_k, \text{origin}_k)])\)
- 1. Stuples = \( \emptyset \); Ttuples = \( \emptyset \)
- 2. For all \( (x_i, \text{origin}_i) \) in input list do
  - 1. If \( \text{origin}_i = "S" \) then Stuples = Stuples \( \cup \{x_i\} \)
  - 2. Else Ttuples = Ttuples \( \cup \{x_i\} \)
- 3. joinResult = MyFavoriteJoinAlg( Stuples, Ttuples )
- 4. Output joinResult

Remaining Challenges

What is the best way to cover all true-valued cells?

And how do we know which matrix cells have value true?

Why Randomization?

- Avoids pre-processing step to assign row/column IDs to records
- Effectively removes output skew
- Input sizes very close to target
  - Chernoff bound: due to large number of records per reducer, probability of receiving 10% or more over target is virtually zero
- Side-benefit: join matrix does not have to have \(|S| \) by \(|R| \) cells, could be much smaller!

Cartesian Product Computation

- Start with cross-product \(S \times T\)
  - Entire matrix needs to be covered by \( r \) reducer regions
- Lemma 1: use square-shaped regions!
  - A reducer that covers \( c \) cells of join matrix \( M \) will receive at least \( 2 \sqrt{c} \) input tuples
Optimal Cover for M

- Need to cover all $|S| \cdot |T|$ matrix cells
  - Lower bound for max-reducer-output: $|S| \cdot |T|/r$
  - Lemma 1 implies lower bound for max-reducer-input: $2\sqrt{r} \cdot |S| \cdot |T|/r$
- Can we match these lower bounds?
  - YES: Use $r$ squares, each $\sqrt{|S| \cdot |T|/r}$ cells wide/tall
- Can this be achieved for given $S$, $T$, $r$?

Easy Case

- $|S|$, $|T|$ are both multiples of $\sqrt{|S| \cdot |T|/r}$
  - Optimal!

![Optimal square region](image)
![Join matrix (cross-product)](image)

Also Easy

- $|S| < |T|/r$
  - Implies $|S| < \sqrt{r} \cdot |S| \cdot |T|/r$
  - Lower bound for input not achievable
- Optimal: use rectangles of size $|S|$ by $|T|/r$

![“Idealistic” square region](image)
![Actual optimal region](image)

Hard Case

- $|T|/r \leq |S| \leq |T|$ and at least one is not multiple of $\sqrt{|S| \cdot |T|/r}$

![Optimal square region](image)
![9 regions: - 6 fit - 3 do not fit](image)

Solution For Hard Case

- “Inflate” squares until they just cover the matrix
  - Worst case: only one square did fit initially, but leftover just too small to fit more rows or columns

![Need to at most double side-length of optimal square](image)

Near-Optimality For Cross-Product

- Every region has less than $4 \cdot \sqrt{r} \cdot |S| \cdot |T|/r$ input records
  - Lower bound: $2 \sqrt{r} \cdot |S| \cdot |T|/r$
- Every region contains less than $4 \cdot |S| \cdot |T|/r$ cells
  - Lower bound: $|S| \cdot |T|/r$
- Summary: max-reducer-input and max-reducer-output are within a factor of 2 and 4 of the lower bound, respectively
  - Usually much better: if 10 by 10 squares fit initially, they are within a factor of 1.1 and 1.21 of lower bound!
From Cross-Product To Joins

- Near-optimality only shown for cross-product
- Randomization of 1-Bucket-Theta tends to distribute output very evenly over regions
  - Join-specific mapping unlikely to improve max-reducer-output significantly
  - 1-Bucket-Theta wins for output-size dominated joins
- Join-specific mapping has to beat 1-Bucket-Theta on input cost!
  - Avoid covering empty matrix regions

Finding Empty Matrix Regions

- For a given matrix region, prove that it contains no join result
- Need statistics about S and T
- Need simple enough join predicate
  - Histogram bucket: $S.A > 8 \land T.A < 7$
  - Join predicate: $S.A = T.A$
  - Easy to show that bucket property implies negation of join predicate
- Not possible for “blackbox” join predicates

Approximate Join Matrix

- For a given matrix region, prove that it contains no join result
- Need statistics about S and T
- Need simple enough join predicate
  - Histogram bucket: $S.A > 8 \land T.A < 7$
  - Join predicate: $S.A = T.A$
  - Easy to show that bucket property implies negation of join predicate
- Not possible for “blackbox” join predicates

What Can We Do?

- Even if we could guess a better algorithm than 1-Bucket-Theta, we cannot use it unless we can prove that it does not miss any join results
- Can do this for many popular join types
  - Equi-join: $S.A = T.A$
  - Inequality-join: $S.A \leq T.A$
  - Band-join: $R.A - \varepsilon_1 \leq S.A \leq R.A + \varepsilon_2$
- Need histograms (easy and cheap to compute)

M-Bucket-I

- Uses multiple-bucket histograms to minimize max-reducer-input
- First identifies candidate cells
- Then tries to cover all candidate cells with r regions
  - Binary search over max-reducer-input values
    - Min: $2 \cdot \sqrt{\text{#candidateCells}} / r$; max: $|S| + |T|$
  - Works on block of consecutive rows
    - Find “best” block (most candidate cells covered per region)
    - Continue with next block, until all candidate cells covered, or running out of regions

M-Bucket-I Illustration

- Uses multiple-bucket histograms to minimize max-reducer-input
- First identifies candidate cells
- Then tries to cover all candidate cells with r regions
  - Binary search over max-reducer-input values
    - Min: $2 \cdot \sqrt{\text{#candidateCells}} / r$; max: $|S| + |T|$
  - Works on block of consecutive rows
    - Find “best” block (most candidate cells covered per region)
    - Continue with next block, until all candidate cells covered, or running out of regions

- Best: 1
- Max input = 3
- And so on.
M-Bucket-O

- Similar to M-Bucket-I, but tries to minimize max-reducer-output
- Binary search over max-reducer-output values
- Problem: estimate number of result cells in regions inside a histogram bucket
  - Estimate can be poor, even for fine-grained histogram
  - Input-size estimation much more accurate than output-size estimation

Extension: Memory-Awareness

- Input for region might exceed reducer memory
- Solutions
  - Use I/O-based join implementation in Reduce, or
  - Create more (and hence smaller) regions
- 1-Bucket-Theta: use squares of side-length Mem/2
- M-Bucket-I: Instead of binary search on max-reducer-input, set it immediately to Mem
- Similar for M-Bucket-O

Experiments: Basic Setup

- 10-machine cluster
  - Quad-core Xeon 2.4GHz, 8MB cache, 8GB RAM, two 250GB 7.2K RPM hard disks
- Hadoop 0.20.2
  - One machine head node, other nine worker nodes
  - One Map or Reduce task per core
  - DFS block size of 64MB
  - Data stored on all 10 machines

Data Sets

- Cloud
  - Cloud reports from ships and land stations
  - 382 million records, 28 attributes, 28.8GB total size
- Cloud-5-1, Cloud-5-2
  - Independent random samples from Cloud, each with 5 million records
- Synth-α
  - Pair of data sets of 5 million records each
  - Record is single integer between 1 and 1000
  - Data set 1: uniformly generated
  - Data set 2: Zipf distribution with parameter α
    - For α=0, data is perfectly uniform

Skew Resistance: Equi-Join

- 1-Bucket-Theta vs. standard equi-join algorithm
- Output-size dominated join
  - Max-reducer-output determines runtime

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Output size (billion)</th>
<th>1-Bucket-Theta</th>
<th>Standard algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Output imbalance</td>
<td>Runtime (sec)</td>
</tr>
<tr>
<td>Synth-0</td>
<td>25.00</td>
<td>1.0030</td>
<td>857</td>
</tr>
<tr>
<td>Synth-0.4</td>
<td>24.99</td>
<td>1.0023</td>
<td>650</td>
</tr>
<tr>
<td>Synth-0.6</td>
<td>24.98</td>
<td>1.0033</td>
<td>676</td>
</tr>
<tr>
<td>Synth-0.8</td>
<td>24.95</td>
<td>1.0068</td>
<td>678</td>
</tr>
<tr>
<td>Synth-1</td>
<td>24.91</td>
<td>1.0089</td>
<td>667</td>
</tr>
</tbody>
</table>

Selective Band-Join

```
SELECT S.date, S.longitude, S.latitude, T.latitude
FROM Cloud AS S, Cloud AS T
WHERE S.date = T.date
AND S.longitude = T.longitude AND
ABS(S.latitude - T.latitude) <= 10
```

- 390M output vs. 764M input records
- M-Bucket-I for different histogram granularities
M-Bucket-I Results

- 10-run averages (stdev < 15%)

M-Bucket-I Details

- M-Bucket-I for 1-bucket histogram is improved version of original 1-Bucket-Theta
  - 1-Bucket-Theta might keep reducers idle
- Out-of-memory for 1-bucket and 100-bucket cases
  - Used memory-aware version of algorithm
  - Creates $c \cdot r$ regions for $r$ reducers for smallest integer $c$ that allows in-memory processing
- Input duplication rate: total mapper output size vs. total mapper input size
  - 31.22, 8.92, 1.93, 1.043, 1.00048, 1.00025 for histograms with 1, 10, 100, 1000, 10K, 100k, and 1M buckets

M-Bucket-O Results

- 10-run averages (stdev < 4%)

Not-So-Selective Band-Join

```
SELECT S.latitude, T.latitude
FROM Cloud-5-1 AS S, Cloud-5-2 AS T
WHERE ABS(S.latitude-T.latitude) <= 2
```

- 22 billion output vs. 10 million input records
- M-Bucket-O for different histogram granularities

M-Bucket-O Details

- M-Bucket-O for 1-bucket histogram is improved version of original 1-Bucket-Theta
- Data set has 5951 distinct latitude values
- Input duplication rate: total mapper output size vs. total mapper input size
  - 7.50, 4.14, 1.46, 1.053, 1.035 for histograms with 1, 10, 100, 1000, and 5951 buckets

## Detailed cost breakdown

<table>
<thead>
<tr>
<th>Step</th>
<th>Number of histogram buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Quantiles</td>
<td>0</td>
</tr>
<tr>
<td>Histogram</td>
<td>0</td>
</tr>
<tr>
<td>Heuristic</td>
<td>74</td>
</tr>
<tr>
<td>Join</td>
<td>49,384</td>
</tr>
<tr>
<td>Total</td>
<td>49,458</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step</th>
<th>Number of histogram buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Quantiles</td>
<td>0</td>
</tr>
<tr>
<td>Histogram</td>
<td>0</td>
</tr>
<tr>
<td>Heuristic</td>
<td>0.04</td>
</tr>
<tr>
<td>Join</td>
<td>1279</td>
</tr>
<tr>
<td>Total</td>
<td>1279</td>
</tr>
</tbody>
</table>
Summary

- Join model for creation and reasoning about parallel algorithms
- Near-optimal randomized algorithm for output-size dominated joins
- Improved heuristics for popular very selective joins

Future Directions

- Explore broader model applicability
  - Very general model
  - Works for size-skewed joins where one set fits in memory
    - Improves completion time of Map-only implementation
  - Algorithm can be executed sequentially
    - Can tune it to available memory
- Multi-way theta-joins
- Optimizer to select best implementation for given join problem