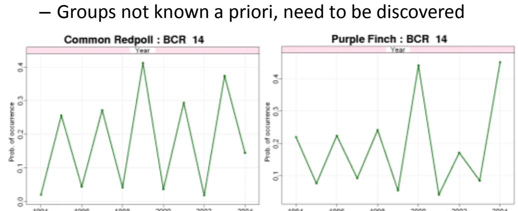
Now that we have seen important design patterns and MapReduce algorithms for simpler problems, let's look at some more complex problems, starting with general joins.

Joins in MapReduce

- Data sets $S=\{s_1,\ldots, s_{|S|}\}$ and $T=\{t_1,\ldots, t_{|T|}\}$
- Find all pairs (s_i, t_j) that satisfy some predicate
- Examples
  - Pairs of similar or complementary function summaries
  - Facebook and Twitter posts by same user or from same location
- Typical goal: minimize job completion time

Function-Join Pattern

- Find groups of summaries with certain properties of interest
  - Similar trends, opposite trends, correlations
  - Groups not known a priori, need to be discovered



Existing Join Support

- Hadoop has some built-in join support, but our goal is to design our own algorithms
  - Built-in support is limited
  - We want to understand important algorithm design principles
- "Join" usually just means equi-join, but we also want to support other join predicates
- Note: recall join discussion from earlier lecture

Joining Large With Small

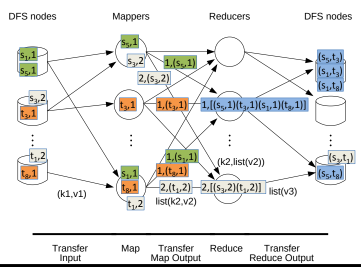
- Assume data set T is small enough to fit in memory
- Can run Map-only join
  - Load T onto every mapper
  - Map: join incoming S-tuple with T, output all matching pairs
    - Can scan entire T (nested loop) or use index on T (index nested loop)
- Downside: need to copy T to all mappers
  - Not so bad, since T is small

Distributed Cache

- Efficient way to copy files to all nodes processing a certain task
  - Use it to send small T to all mappers
- Part of the job configuration
- Hadoop still needs to move the data to the worker nodes, so use this with care
  - But it avoids copying the file for every task on the same node

Recall: Standard Equi-Join Algorithm

- Join condition: S.A=T.A
- Map(s) = (s.A, s); Map(t) = (t.A, t)
- Reduce combines S-tuples and T-tuples with same key

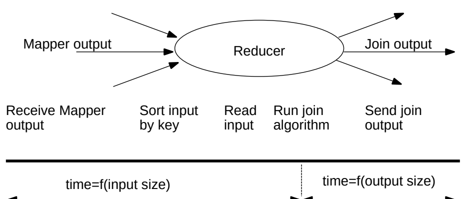


Problems With Standard Approach

- Degree of parallelism limited by number of distinct A-values
- Data skew
  - If one A-value dominates, reducer processing that key will become bottleneck
- Does not generalize to other joins

Reducer-Centric Cost Model

- Difference between join implementations starts with Map output

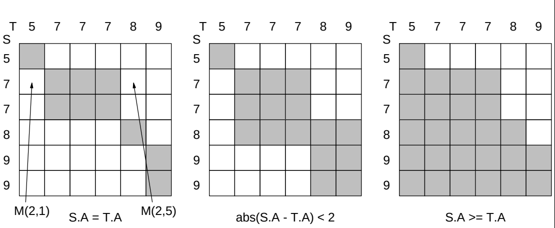


Optimization Goal: Minimal Job Completion time

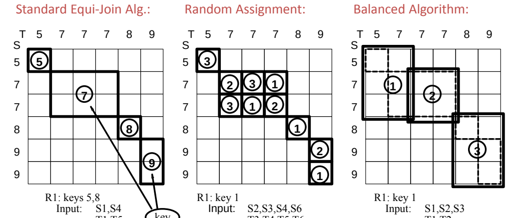
- Assume all reducers are similarly capable
- Processing time at reducer is approximately monotonic in input and output size
- Hence need to minimize max-reducer-input or max-reducer-output
- Join problem classification
  - Input-size dominated: minimize max-reducer-input
  - Output-size dominated: minimize max-reducer-output
  - Input-output balanced: minimize combination of both

Join Model

- Join-matrix M: M(i, j) = *true*, if and only if (s_i, t_j) in join result
- Cover each *true*-valued cell by exactly one reducer



Standard Equi-Join Alg.: | Random Assignment: | Balanced Algorithm:



Standard Equi-Join Alg.:

R1: keys 5,8
Input: S1,S4 T1,T5
Output: 2 tuples

R2: key 7
Input: S2,S3 T2,T3,T4
Output: 6 tuples

R3: key 9
Input: S5,S6 T6
Output: 2 tuples

max-reducer-input = 5
max-reducer-output = 6

Random Assignment:

R1: key 1
Input: S2,S3,S4,S6 T3,T4,T5,T6
Output: 4 tuples

R2: key 2
Input: S2,S3,S5 T2,T4,T6
Output: 3 tuples

R3: key 3
Input: S1,S2,S3 T1,T2,T3
Output: 3 tuples

max-reducer-input = 8
max-reducer-output = 4

Balanced Algorithm:

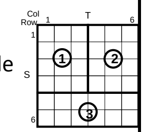
R1: key 1
Input: S1,S2,S3 T1,T2
Output: 3 tuples

R2: key 2
Input: S2,S3 T3,T4
Output: 4 tuples

R3: key 3
Input: S4,S5,S6 T5,T6
Output: 3 tuples

max-reducer-input = 5
max-reducer-output = 4

1-Bucket-Random: Map



- Input: tuple $x \in S \cup T$, matrix-to-reducer mapping *lookup* table

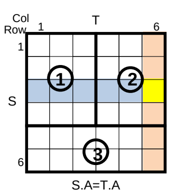
1. If $x \in S$ then
   1. matrixRow = random(1, |S|)
   2. Forall regionID in lookup.getRegions(matrixRow)
      1. Output (regionID, (x, "S"))
2. Else
   1. matrixCol = random(1, |T|)
   2. Forall regionID in lookup.getRegions(matrixCol)
      1. Output (regionID, (x, "T"))

1-Bucket-Random: Reduce

- Input: (ID, [(x_1, origin_1),…, (x_k, origin_k)])

1. Stuples = $\emptyset$; Ttuples = $\emptyset$
2. Forall (x_i, origin_i) in input list do
   1. If origin_i = "S" then Stuples = Stuples $\cup$ {x_i}
   2. Else Ttuples = Ttuples $\cup$ {x_i}
3. joinResult = MyFavoriteJoinAlg(Stuples, Ttuples)
4. Output joinResult

1-Bucket-Random Example



Map:

Input tuple	Random row/col	Output
S1.A=5	3	(1,S1),(2,S1)
S2.A=7	5	(3,S2)
S3.A=7	1	(1,S3),(2,S3)
S4.A=8	5	(3,S4)
S5.A=9	1	(1,S5),(2,S5)
S6.A=9	2	(1,S6),(2,S6)
T1.A=5	6	(2,T1),(3,T1)
T2.A=7	2	(1,T2),(3,T2)
T3.A=7	2	(1,T3),(3,T3)
T4.A=7	3	(1,T4),(3,T4)
T5.A=8	6	(2,T5),(3,T5)
T6.A=9	4	(2,T6),(3,T6)

Reduce:

Reducer X: key 1
Input: S1, S3, S5 ,S6 T2, T3, T4
Output: (S3,T2),(S3,T3),(S3,T4)

Reducer Y: key 2
Input: S1, S3, S5, S6 T1, T5, T6
Output: (S1,T1),(S5,T6),(S6,T6)

Reducer Z: key 3
Input: S2, S4 T1, T2, T3, T4, T5, T6
Output: (S2,T2),(S2,T3), (S2,T4),(S4,T5)

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!

Remaining Challenges

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

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

Cartesian Product Computation

- Start with cross-product $S \times T$
  - Entire matrix needs to be covered by r reducer regions (= r reduce tasks)
- 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|·|T| matrix cells
  - Lower bound for max-reducer-output: |S|·|T|/r
  - Lemma 1 implies lower bound for max-reducer-input: 2·sqrt(|S|·|T|/r)
- Can we match these lower bounds?
  - YES: Use r squares, each sqrt(|S|·|T|/r) cells wide/tall
- Can this be achieved for given S, T, r?

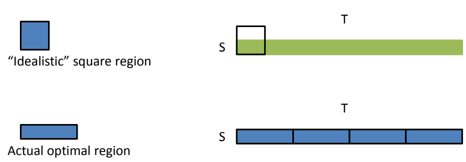
Easy Case

- |S|, |T| are both multiples of sqrt(|S|·|T|/r)
- Optimal!

Optimal square region
Join matrix (cross-product)

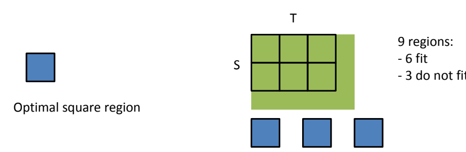
Also Easy

- |S| < |T|/r
  - Implies |S| < sqrt(|S|·|T|/r)
  - Lower bound for input not achievable
- Optimal: use rectangles of size |S| by |T|/r



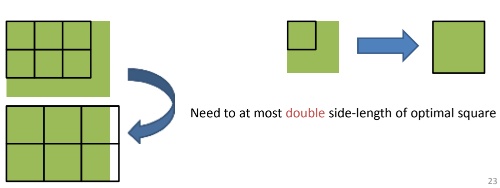
Hard Case

- |T|/r ≤ |S| ≤ |T| and at least one is not multiple of sqrt(|S|·|T|/r)



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



Near-Optimality For Cross-Product

- Every region has less than 4·sqrt(|S|·|T|/r) input records
  - Lower bound: 2·sqrt(|S|·|T|/r)
- Every region contains less than 4·|S|·|T|/r cells
  - Lower bound: |S|·|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!

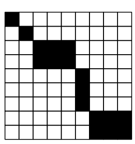
From Cross-Product To Joins

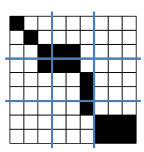
- Near-optimality shown for cross-product
- Randomization of 1-Bucket-Random tends to distribute output very evenly over regions
  - Join-specific mapping unlikely to improve max-reducer-*output* significantly
  - 1-Bucket-Random wins for any output-size dominated join
- Join-specific mapping has to beat 1-Bucket-Random on input cost: avoid covering empty matrix regions

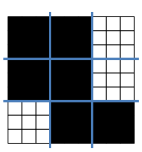
Finding Empty Matrix Regions

- For a given matrix region, prove that it contains no join result
- Need statistics about S and T and a simple enough join predicate
  - Histogram bucket: S.A > 8 $\wedge$ 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


True join matrix


Histogram boundaries


Candidate cells to be covered by algorithm

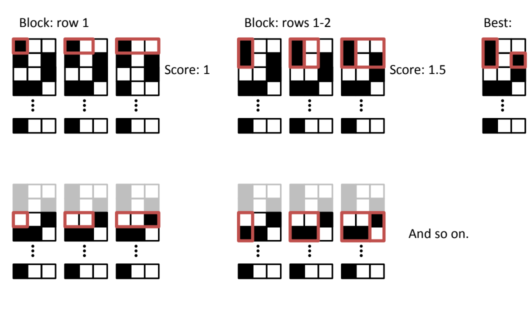
What Can We Do?

- Proving buckets to be empty is easy 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 + ε_2
- For statistics, use histograms
  - Two 1-dimensional histograms: one on S the other on T
  - Easy and cheap to compute

M-Bucket-I

- Uses **M**ultiple-bucket histograms to minimize max-reducer-**I**nput
- First identifies candidate cells, then tries to cover all candidate cells with r regions
  - Binary search over max-reducer-input values
    - Min: 2·sqrt(#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



M-Bucket-O

- Similar to M-Bucket-I, but tries to minimize max-reducer-**O**utput
- Binary search over max-reducer-output values
- Problem: needs to 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-Random: 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-Random vs. standard equi-join algorithm
- Output-size dominated join
  - Max-reducer-output determines runtime

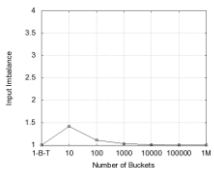
		1-Bucket-Random		Standard algorithm	
Data Set	Output size (billion)	Output imbalance	Runtime (secs)	Output Imbalance	Runtime (secs)
Synth-0	25.00	1.0030	657	1.001	701
Synth-0.4	24.99	1.0023	650	1.254	722
Synth-0.6	24.98	1.0033	676	1.778	923
Synth-0.8	24.95	1.0068	678	3.010	1482
Synth-1	24.91	1.0089	667	5.312	2489

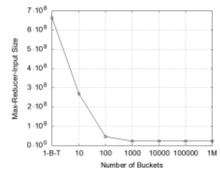
Selective Band-Join

```
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
```

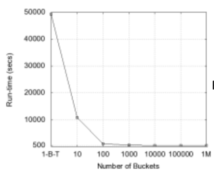
- 390M output vs. 764M input records
- M-Bucket-I for different histogram granularities

M-Bucket-I Results





10-run averages
(stdev < 15%)



Runtime for MapReduce only!

M-Bucket-I Details

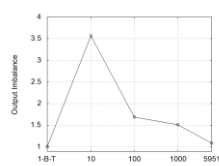
- M-Bucket-I for 1-bucket histogram is improved version of original 1-Bucket-Random
  - 1-Bucket-Random might keep reducers idle
- Out-of-memory for 1-bucket and 100-bucket cases
  - Used memory-aware version of algorithm
  - Creates c·r regions for r reducers for smallest integer c that allows in-memory 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

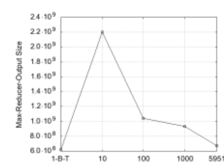
Not-So-Selective Band-Join

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

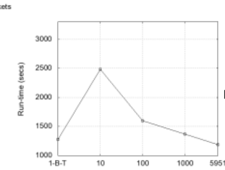
- 22 billion output vs. 10 million input records
- M-Bucket-O for different histogram granularities

M-Bucket-O Results





10-run averages
(stdev < 4%)



Runtime for MapReduce only!

M-Bucket-O Details

- M-Bucket-O for 1-bucket histogram is improved version of original 1-Bucket-Random
- 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

M-Bucket-I on Cloud data set (input-size dominated join):

Step	Number of histogram buckets						
	1	10	100	1000	10,000	100,000	1,000,000
Quantiles	0	115	120	117	122	124	122
Histogram	0	140	145	147	157	167	604
Heuristic	74	9	0.8	1.5	17	118	111
Join	49,384	10,905	1157	595	548	540	536
Total	49,458	11169	1423	861	844	949	1373

M-Bucket-O on Cloud-5 data sets (output-size dominated join):

Step	Number of histogram buckets				
	1	10	100	1000	5951
Quantiles	0	4.5	4.5	4.8	4.9
Histogram	0	26.2	25.8	25.6	25.6
Heuristic	0.04	0.04	0.05	0.24	0.81
Join	1279	2483	1597	1369	1188
Total	1279	2514	1627	1399	1219

Detailed cost breakdown

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

- Multi-way theta-joins
- Optimizer to select best implementation for given join problem
- Consider other optimization goals