MapReduce Algorithms

Join and Multiway Join
Tyranny of Communication Cost
Size/Communication Tradeoffs
Application to Similarity Join, Matrix Multiplication

Jeffrey D. Ullman
Stanford University
Roughly: wall-clock time is inversely proportional to number of processors.

In map-reduce context: mappers produce one (or a few) key-value pairs per input, independent of the number of different reducers.

2-way join is embarrassingly parallel; multiway join is not.
Given: a collection of relations, each with attributes labeling their columns.

Find: Those tuples over all the attributes such that when restricted to the attributes of any relation \( R \), that tuple is in \( R \).
Example: Natural Join

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Join of $R(A,B)$ with $S(B,C)$ is the set of tuples $(a,b,c)$ such that $(a,b)$ is in $R$ and $(b,c)$ is in $S$.

Mappers need to send $R(a,b)$ and $S(b,c)$ to the same reducer, so they can be joined there.

**Mapper output**: key = B-value, value = relation and other component (A or C).

**Example**: $R(1,2) \rightarrow (2, (R,1))$

$S(2,3) \rightarrow (2, (S,3))$
Mapping Tuples

Mapper for R(1,2) → (2, (R,1))

Mapper for R(4,2) → (2, (R,4))

Mapper for S(2,3) → (2, (S,3))

Mapper for S(5,6) → (5, (S,6))
Grouping Phase

- There is a reducer for each key.
  - **Note**: “reducer” != “Reduce task.”
  - A *reducer* is simply a key and its associated values.
- Every key-value pair generated by any mapper is sent to the reducer for its key.
Mapping Tuples

Mapper for $R(1,2)$

(2, (R,1))

Mapper for $R(4,2)$

(2, (R,4))

Mapper for $S(2,3)$

(2, (S,3))

Mapper for $S(5,6)$

(5, (S,6))

Reducer for $B = 2$

Reducer for $B = 5$
As a Key + List of Values

(2, [(R,1), (R,4), (S,3)]) → Reducer for B = 2

(5, [(S,6)]) → Reducer for B = 5
Given key $b$ and a list of values that are either $(R, a_i)$ or $(S, c_j)$, output each triple $(a_i, b, c_j)$.

- Thus, the number of outputs made by a reducer is the product of the number of R’s on the list and the number of S’s on the list.
Output of the Reducers

(2, [(R,1), (R,4), (S,3)]) → Reducer for B = 2 → (1,2,3), (4,2,3)

(5, [(S,6)]) → Reducer for B = 5 →
Consider a chain of three relations:
\[ R(A, B) \ JOIN \ S(B, C) \ JOIN \ T(C, D) \]

**Example**: R, S, and T are “friends” relations.

We could join any two by the 2-way map-reduce algorithm, then join the third with the resulting relation.

But intermediate joins are large.
3-Way Join – (2)

- An alternative is to divide the work among \( k = m^2 \) Reduce tasks.
- Hash both B and C to \( m \) buckets.
- A Reduce task corresponds to a hashed B-value and a hashed C-value.
  - And keys must be only the hash values, not the exact B- and C-values.
3-Way Join – (3)

- Each S-tuple S(b,c) is sent to one Reduce task: (h(b), h(c)).
- But each tuple R(a,b) must be sent to \( m \) Reduce tasks (h(b), x).
- And each tuple T(c,d) must be sent to \( m \) Reduce tasks (y, h(c)).
Example: $m = 4$; $k = 16$. 

```
h(b) = 0
h(b) = 1
h(b) = 2
h(b) = 3
...

s(b, c)  

R(a, b)  

T(c, d)  

h(c) = 0 1 2 3
```
Thus, any joining tuples $R(a,b)$, $S(b,c)$, and $T(c,d)$ will be joined at the Reduce task $(h(b), h(c))$.

**Communication cost**: $s + mr + mt$.

- **Convention**: Lower-case letter is the size of the relation whose name is the corresponding upper-case letter.
- **Example**: $r$ is the size of $R$. 
Suppose for simplicity that:
- Relations R, S, and T have the same size r.
- The probability of two tuples joining is p.

The 3-way join has communication cost \( r(2m + 1) \).

Two two-way joins have a cost of:
- 3r to read the relations, plus
- \( pr^2 \) to read the join of the first two.
- Total = \( r(3 + pr) \).
3-way beats 2-way if $2m+1 < 3+pr$.

$pr$ is the multiplicity of each join.

- Thus, the 3-way chain-join is useful when the multiplicity is high.

**Example**: relations are “friends”; $pr$ is about 300. $m^2 = k$ can be 20,000.

**Example**: relations are Web links; $pr$ is about 15. $m^2 = k$ can be 64.
Some Questions

- When we discussed the 3-way chain-join \( R(A, B) \ JOIN S(B, C) \ JOIN T(C,D) \), we used attributes B and C for the \textit{map-key} (index for the Reduce tasks).
- Why not include A and/or D?
- Why use the same number of buckets for B and C?
For the general problem, we use a *share variable* for each attribute.

- The number of buckets into which values of that attribute are hashed.

**Convention**: The share variable for an attribute is the corresponding lower-case letter.

- **Example**: the share variable for attribute A is always $a$. 
The product of all the share variables is $k$, the number of reducers.
The communication cost for a multiway join is the sum of:
- The size of each relation times the
- Product of the share variables for the attributes that *do not* appear in the schema of that relation.
Consider the cyclic join
\[ R(A, B) \Join S(B, C) \Join T(A, C) \]
Cost function is \( rc + sa + tb \).
Construct the Lagrangean:
\[ rc + sa + tb - \lambda(abc - k) \]
Take derivative with respect to each share variable, then multiply by that variable.
- Result is 0 at minimum.
Example – Continued

- d/da of \( rc + sa + tb - \lambda(abc - k) \) is \( s - \lambda bc \).
- Multiply by \( a \) and set to 0: \( sa - \lambda abc = 0 \).
  - **Note**: \( abc = k \): \( sa = \lambda k \).
- Similarly, d/db and d/dc give: \( sa = tb = rc = \lambda k \).
- **Solution**: \( a = (krt/s^2)^{1/3} \); \( b = (krs/t^2)^{1/3} \); \( c = (kst/r^2)^{1/3} \).
- Communication cost = \( rc + sa + tb = 3(krst)^{1/3} \).
Dominated Attributes

- Certain “dominated” attributes can’t be in the map-key.
- A *dominates* B if every relation of the join with B also has A.
- **Example:**

  \[ R(A, B, C) \ JOIN \ S(A, B, D) \ JOIN \ T(A, E) \ JOIN \ U(C, E) \]

  Every relation with B

  Also has A
Example – (2)

R(A,B,C) JOIN S(A,B,D) JOIN T(A,E) JOIN U(C,E)

- Cost expression:
  \[ rde + sce + tbcde + uabcd \]
- Since \( b \) appears wherever \( a \) does, if there were a minimum-cost solution with \( b > 1 \), we could replace \( b \) by 1 and \( a \) by \( ab \), and the cost would lower.
This rule explains why, in the discussion of the chain join

\[ R(A, B) \text{ JOIN } S(B, C) \text{ JOIN } T(C,D) \]

we did not give dominated attributes A and D a share.

**Note:** Dominated attributes helps, but the solution to the Lagrangean equations in the general case is complex.
A **star join** combines a huge *fact table* $F(A_1, A_2, \ldots, A_n)$ with large but smaller *dimension tables* $D_1(A_1, B_1), D_2(A_2, B_2), \ldots, D_n(A_n, B_n)$.

- There may be other attributes not shown, each belonging to only one relation.
- **Example**: Facts = sales; dimensions tell about buyer, product, etc.
- Used for analytic queries: join the fact table with selections and aggregations on some of the dimensions.
Star-Join Pattern

\[ \text{Diagram with nodes } A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4 \]
Map-key = the A’s.
- B’s are dominated.

**Solution:** \( d_i / a_i = \lambda k \) for all \( i \).
- That is, the shares are proportional to the dimension-table sizes.
Cool Application of Star Join Result

- **Shard** the fact table among compute nodes permanently.
- Replicate needed pieces of dimension tables.
- What is the best way to shard?
Example: Shard by Product

But all Customer tuples for customers who bought shirts, or **might** buy shirts go here.

- Shirts
- Pants
- Pots and Pans

Only Item tuples for shirts go here

Only Item tuples for pants go here
Our solution lets you shard the fact table to $k$ nodes in a data-independent way.

Replication of tuples in the dimension tables is minimized.
At a conference I attended in March, 2012, a paper from a major software/hardware vendor described a system that:

- Shards the Fact table, but
- Fully replicates the dimension tables.
Tyranny of Communication Cost

The Drug Interaction Problem
A Failed Attempt
Lowering the Communication
Data consists of records for 3000 drugs.

- List of patients taking, dates, diagnoses.
- About 1M of data per drug.

Problem is to find drug interactions.

- Example: two drugs that when taken together increase the risk of heart attack.

Must examine each pair of drugs and compare their data.
The first attempt used the following plan:

- **Key** = set of two drugs \( \{i, j\} \).
- **Value** = the record for one of these drugs.

Given drug \( i \) and its record \( R_i \), the mapper generates all key-value pairs \( (\{i, j\}, R_i) \), where \( j \) is any other drug besides \( i \).

Each reducer receives its key and a list of the two records for that pair: \( (\{i, j\}, [R_i, R_j]) \).
Example: Three Drugs

Mapper for drug 1

{1, 2} Drug 1 data

{1, 3} Drug 1 data

Mapper for drug 2

{1, 2} Drug 2 data

{2, 3} Drug 2 data

Mapper for drug 3

{1, 3} Drug 3 data

{2, 3} Drug 3 data

Reducer for {1, 2}

Reducer for {1, 3}

Reducer for {2, 3}
Example: Three Drugs

Mapper for drug 1

{1, 2} Drug 1 data

{1, 3} Drug 1 data

Mapper for drug 2

{1, 2} Drug 2 data

{2, 3} Drug 2 data

Mapper for drug 3

{1, 3} Drug 3 data

{2, 3} Drug 3 data

Reducer for {1,2}

Reducer for {1,3}

Reducer for {2,3}
Example: Three Drugs

{1, 2} Drug 1 data Drug 2 data Reducer for \{1,2\}

{1, 3} Drug 1 data Drug 3 data Reducer for \{1,3\}

{2, 3} Drug 2 data Drug 3 data Reducer for \{2,3\}
What Went Wrong?

- 3000 drugs
- times 2999 key-value pairs per drug
- times 1,000,000 bytes per key-value pair
- = 9 terabytes communicated over a 1Gb Ethernet
- = 90,000 seconds of network use.
Suppose we group the drugs into 30 groups of 100 drugs each.

- Say $G_1 =$ drugs 1-100, $G_2 =$ drugs 101-200,..., $G_{30} =$ drugs 2901-3000.
- Let $g(i) =$ the number of the group into which drug $i$ goes.
The Map Function

- A key is a set of two group numbers.
- The mapper for drug $i$ produces 29 key-value pairs.
  - Each key is the set containing $g(i)$ and one of the other group numbers.
  - The value is a pair consisting of the drug number $i$ and the megabyte-long record for drug $i$. 
The Reduce Function

- The reducer for pair of groups \( \{m, n\} \) gets that key and a list of 200 drug records – the drugs belonging to groups \( m \) and \( n \).
- Its job is to compare each record from group \( m \) with each record from group \( n \).
  - **Special case**: also compare records in group \( n \) with each other, if \( m = n+1 \) or if \( n = 30 \) and \( m = 1 \).
- Notice each pair of records is compared at exactly one reducer, so the total computation is not increased.
The New Communication Cost

- The big difference is in the communication requirement.
- Now, each of 3000 drugs’ 1MB records is replicated 29 times.
  - Communication cost = 87GB, vs. 9TB.
Cost of Map-Reduce Jobs

- On a public cloud, you pay for computation and you also pay for communication.
  - Balancing the two is an important part of algorithm design.
- But you also want the job to finish fast, which requires a high degree of parallelism.
- Often, there is a second trade-off, with high parallelism pushing the communication higher than you would like for minimum cost.
A Picture of the Trade-Offs

Perhaps this, if the deadline is critical.

Or, if you can compromise, perhaps this point.

But if you want to finish this fast, you may want to pick a Suboptimum point.

If this much time is OK, pick optimum.

[Diagram showing communication cost, computation cost, optimum point, total cost, wall-clock time, and trade-offs.]
For some problems, the computation is the same no matter how you partition the problem. However, in many cases, the big issue is whether a reducer has too much input to operate in main memory.

- To get reducers with small input size, you need a lot of communication.
- Results in a step function of cost when communication gets too low.
A Common Case

![Diagram showing communication cost and computation cost](image)

- **Optimum point**
- **Total cost**
- **Communication cost**
- **Computation Cost**
Theory of Map-Reduce Algorithms

Reducer Size
Replication Rate
Mapping Schemas
Lower Bounds
A Model for Map-Reduce Algorithms

1. A set of *inputs*.
   - **Example**: the drug records.

2. A set of *outputs*.
   - **Example**: One output for each pair of drugs.

3. A many-many relationship between each output and the inputs needed to compute it.
   - **Example**: The output for the pair of drugs \( \{i, j\} \) is related to inputs \( i \) and \( j \).
Example: Drug Inputs/Outputs

Drug 1

Drug 2

Drug 3

Drug 4

Output 1-2

Output 1-3

Output 1-4

Output 2-3

Output 2-4

Output 3-4
Example: Matrix Multiplication
Reducer Size

- Reducer size, denoted q, is the maximum number of inputs that a given reducer can have.
  - I.e., the length of the value list.
- Limit might be based on how many inputs can be handled in main memory.
- Or: make q low to force lots of parallelism.
The average number of key-value pairs created by each mapper is the *replication rate*.  
- Denoted $r$.  
- Represents the communication cost per input.
In our model, inputs and outputs are hypothetical.

That is, they represent all the possible inputs that could exist, and the outputs that might be made.

In any execution of the algorithm, the real inputs are a subset of the hypothetical inputs, and the outputs are whatever can be made from those inputs.

Example: HD1 problem to be discussed.
When not all hypothetical inputs are expected to be present, we can raise $q$ proportionally.

**Example**: If we want no more than one million inputs to any reducer, and we expect 10% of inputs to be present, then we can set $q = 10M$.

**Risk of skew.**
- Input selection may not be random.
- Some reducers may get many more inputs than others.
Suppose we use $g$ groups and $d$ drugs.

- A reducer needs two groups, so $q = \frac{2d}{g}$.
- Each of the $d$ inputs is sent to $g-1$ reducers, or approximately $r = g$.
- Replace $g$ by $r$ in $q = \frac{2d}{g}$ to get $r = \frac{2d}{q}$.

Tradeoff!
The bigger the reducers, the less communication.
What we did gives an upper bound on $r$ as a function of $q$.

A solid investigation of map-reduce algorithms for a problem includes lower bounds.

- Proofs that you cannot have lower $r$ for a given $q$. 
A *mapping schema* for a problem and a reducer size $q$ is an assignment of inputs to sets of reducers, with two conditions:

1. No reducer is assigned more than $q$ inputs.
2. For every output, there is some reducer that receives all of the inputs associated with that output.
   - Say the reducer *covers* the output.
Every map-reduce algorithm has a mapping schema.

The requirement that there be a mapping schema is what distinguishes map-reduce algorithms from general parallel algorithms.
Example: Drug Interactions

- d drugs, reducer size q.
- No reducer can cover more than $q^2/2$ outputs.
- There are $d^2/2$ outputs that must be covered.
- Therefore, we need at least $d^2/q^2$ reducers.
- Each reducer gets q inputs, so replication r is at least $q(d^2/q^2)/d = d/q$.
- Half the r from the algorithm we described.

Inputs per reducer | Number of reducers | Divided by number of inputs
Specific Problems

Hamming Distance

Matrix Multiplication
Given a set of bit strings of length $b$, find all those that differ in exactly one bit.

**Example:** For $b=2$, the inputs are 00, 01, 10, 11, and the outputs are (00,01), (00,10), (01,11), (10,11).

**Theorem:** $r > b / \log_2 q$.

**Note:** good example where we do not expect all inputs to be present.
We can use one reducer for every possible output.
Each input is sent to b reducers (so \( r = b \)).
Each reducer outputs its pair if both its inputs are present, otherwise, nothing.
Algorithm with $q = 2^b$

- Alternatively, we can send all inputs to one reducer.
- No replication (i.e., $r = 1$).
- The lone reducer looks at all pairs of inputs that it receives.
Assume b is even.

Two reducers for each string of length b/2.
  - Call them the \textit{left} and \textit{right} reducers for that string.

String w = xy, where |x| = |y| = b/2, goes to the left reducer for x and the right reducer for y.

If w and z differ in exactly one bit, then they will both be sent to the same left reducer (if they disagree in the right half) or to the same right reducer (if they disagree in the left half).

Thus, \( r = 2; \ q = 2^{b/2}. \)
Algorithms Matching Lower Bound

One reducer for each output

Generalized Splitting

Splitting

All inputs to one reducer

$r = \text{replication rate}$

$q = \text{reducer size}$

$b = \text{reducer size}$

$2^1$ $2^{b/2}$ $2^b$
Matrix Multiplication

One-Job Method
Two-Job Method
Comparison
Assume $n \times n$ matrices $AB = C$.

- $A_{ij}$ is the element in row $i$ and column $j$ of matrix $A$.
  - Similarly for $B$ and $C$.

- $C_{ik} = \sum_j A_{ij} \times B_{jk}$.

- Output $C_{ik}$ depends on the $i^{th}$ row of $A$, that is, $A_{ij}$ for all $j$, and the $k^{th}$ column of $B$, that is, $B_{jk}$ for all $j$. 
Computing One Output Value

Row i

Column k

A

B

C
**Important fact**: If a reducer covers outputs $C_{ik}$ and $C_{fg}$, then it also covers $C_{ig}$ and $C_{fk}$.  

**Why?** This reducer has all of rows $i$ and $f$ of $A$ as inputs and also has all of columns $k$ and $g$ of $B$ as inputs.  

Thus, it has all the inputs it needs to cover $C_{ig}$ and $C_{fk}$.  

**Generalizing**: Each reducer covers all the outputs in the “rectangle” defined by a set of rows and a set of columns of matrix $C$. 
The Responsibility of One Reducer
If a reducer gets \( q \) inputs, it gets \( \frac{q}{n} \) rows or columns.

Maximize the number of outputs covered by making the input “square.”

- I.e., \( \# \text{rows} = \# \text{columns} \).
- \( \frac{q}{2n} \) rows and \( \frac{q}{2n} \) columns yield \( \frac{q^2}{4n^2} \) outputs covered.
Lower Bound on Replication Rate

- Total outputs = $n^2$.
- One reducer can cover at most $q^2/4n^2$ outputs.
- Therefore, $4n^4/q^2$ reducers.
- $4n^4/q$ total inputs to all the reducers, divided by $2n^2$ total inputs = $2n^2/q$ replication rate.
- **Example**: If $q = 2n^2$, one reducer suffices and the replication rate is $r = 1$.
- **Example**: If $q = 2n$ (minimum possible), then $r = n$. 
Matching Algorithm

- Divide rows of the first matrix into g groups of n/g rows each.
- Also divide the columns of the second matrix into g groups of n/g columns each.
- $g^2$ reducers, each with $q = 2n^2/g$ inputs consisting of a group of rows and a group of columns.
- $r = g = 2n^2/q$. 
Picture of One Reducer

\[ \text{n/g} \]

\[ = \]

\[ \text{n/g} \]
A better way: use two map-reduce jobs.

Job 1: Divide both input matrices into rectangles.
- Reducer takes two rectangles and produces partial sums of certain outputs.

Job 2: Sum the partial sums.
For $i$ in $I$ and $k$ in $K$, contribution is $\sum_{j \in J} A_{ij} \times B_{jk}$
Divide the rows of the first matrix A into g groups of n/g rows each.
Divide the columns of A into 2g groups of n/2g.
Divide the rows of the second matrix B into 2g groups of n/2g rows each.
Divide the columns of B into g groups of n/g.
Important point: the groups of columns for A and rows for B must have indices that match.
Reducers for First Job

- Reducers correspond to an n/g by n/2g rectangle in A (with row indices I, column indices J) and an n/2g by n/g rectangle in B (with row indices J and column indices K).
  - Call this reducer (I,J,K).
  - Important point: there is one set of indices J that plays two roles.
    - Needed so only rectangles that need to be multiplied are given a reducer.
2g reducers contribute to this area, one for each J.
Convention: i, j, k are individual rows and/or column numbers, which are members of groups I, J, and K, respectively.

Mappers Job 1:
- $A_{ij} \rightarrow$ key = (I,J,K) for any group K; value = ($A, i, j, A_{ij}$).
- $B_{jk} \rightarrow$ key = (I,J,K) for any group I; value = ($B, j, k, B_{jk}$).

Reducers Job 1: For key (I,J,K) produce
$$x_{ijk} = \sum_{j \in J} A_{ij} \times B_{jk}.$$
Job 2: Details

- Mappers Job 2: $x_{ijk} \rightarrow$ key = (i,k), value = $x_{ijk}$.

- Reducers Job 2: For key (i,k), produce output $C_{ik}$
  
  $= \sum_j x_{ijk}$. 
The two methods (one or two map-reduce jobs) essentially do the same computation.

- Every $A_{ij}$ is multiplied once with every $B_{jk}$.
- All terms in the sum for $C_{ik}$ are added together somewhere, only once.

2 jobs requires some extra overhead of task management.
Comparison: Communication Cost

- **One-job method**: \( r = \frac{2n^2}{q} \); there are \( 2n^2 \) inputs, so total communication = \( \frac{4n^4}{q} \).
- **Two-job method** with parameter \( g \):
  - Job 2: Communication = \( (2g)(\frac{n^2}{g^2})(g^2) = 2n^2g \).

Number of reducers contributing to each output

Number of output squares

Area of each square
Job 1 communication:
- $2n^2$ input elements.
- Each generates $g$ key-value pairs.
- So another $2n^2g$.
- Total communication = $4n^2g$.

Reducer size $q = (2)(n^2/2g^2) = n^2/g^2$.
- So $g = n/\sqrt{q}$.
- Total communication = $4n^3/\sqrt{q}$.
  - Compares favorably with $4n^4/q$ for the one-job approach.