

Theory of MapReduce Algorithms

An Example from CS341

Abstractions: Input/Output Mappings,
Mapping Schemas

Reducer-Size/Communication Tradeoffs

Jeffrey D. Ullman
Stanford University/Infolab



The All-Pairs Problem

Motivation: Drug Interactions
A Failed Attempt
Lowering the Communication

The Drug-Interaction Problem

- A real story from CS341 data-mining project class.
- Students involved did a wonderful job, got an “A.”
- But their first attempt at a MapReduce algorithm caused them problems and led to the development of an interesting theory.

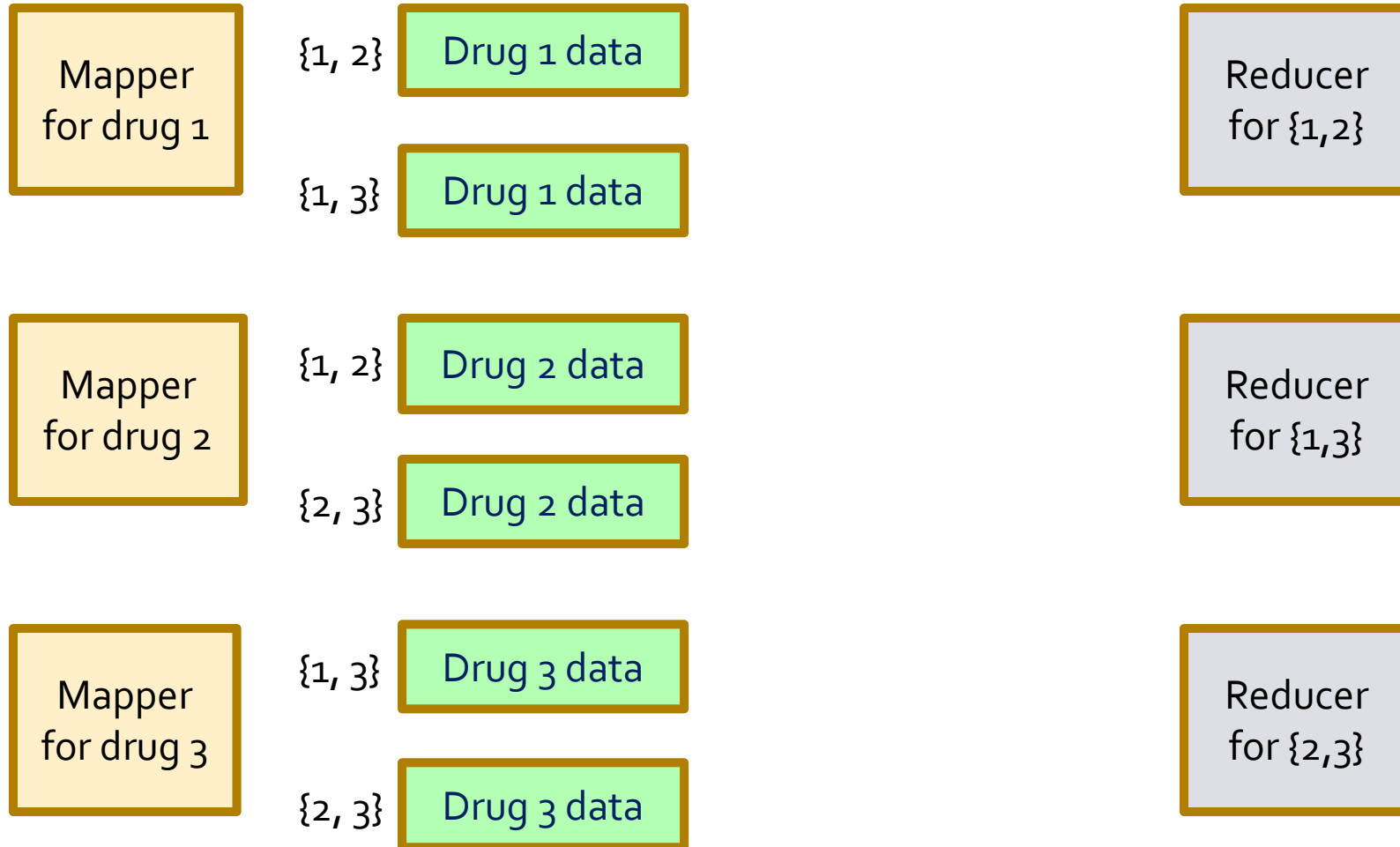
The Drug-Interaction Problem

- Data consisted of records for 3000 drugs.
 - List of patients taking, dates, diagnoses.
 - About 1M of data per drug.
- Problem was 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.

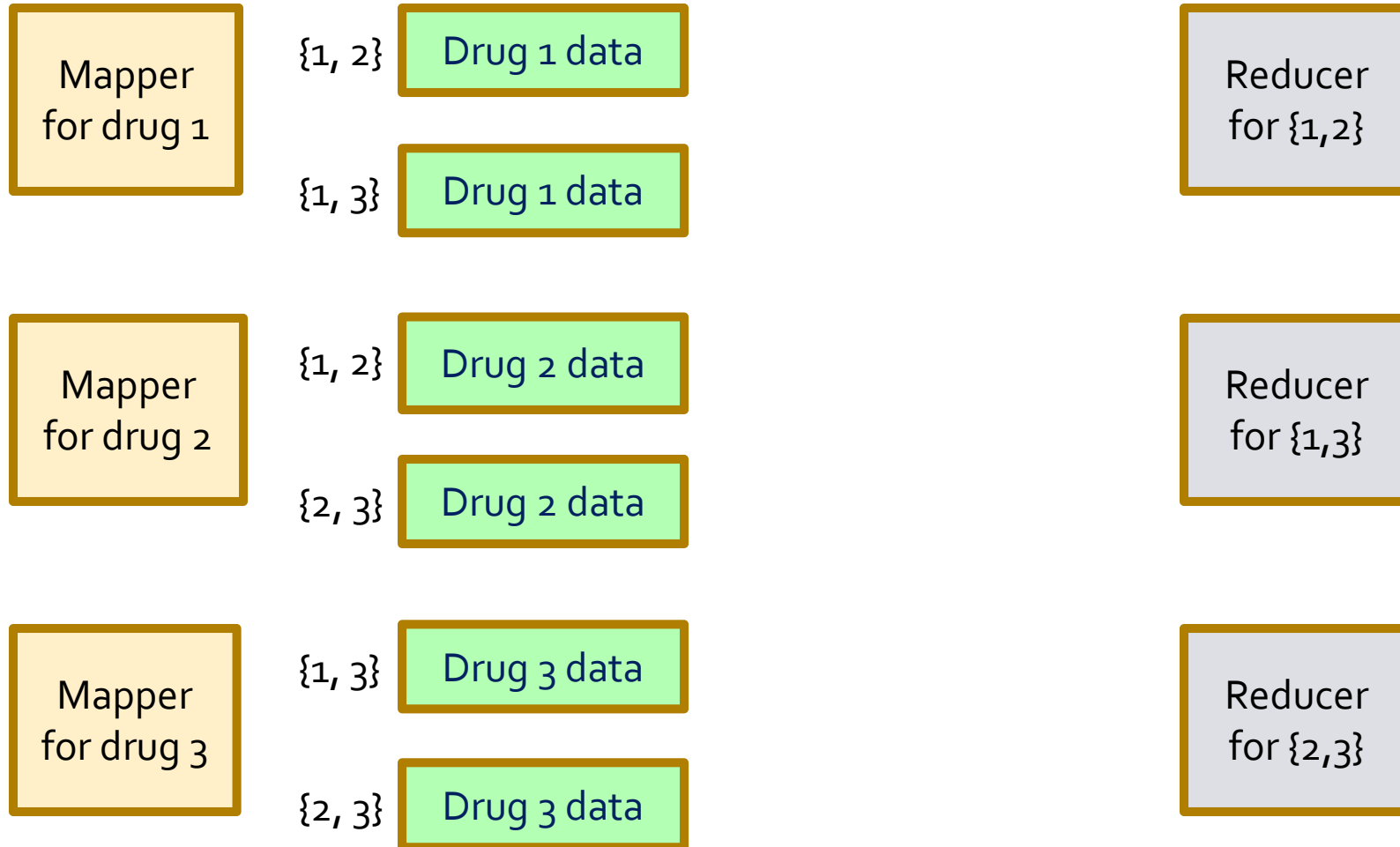
Initial MapReduce Algorithm

- 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: $(\{i, j\}, [R_i, R_j])$.
- **Question for thought:** wouldn't it be better if the value were a pointer to R_i ?

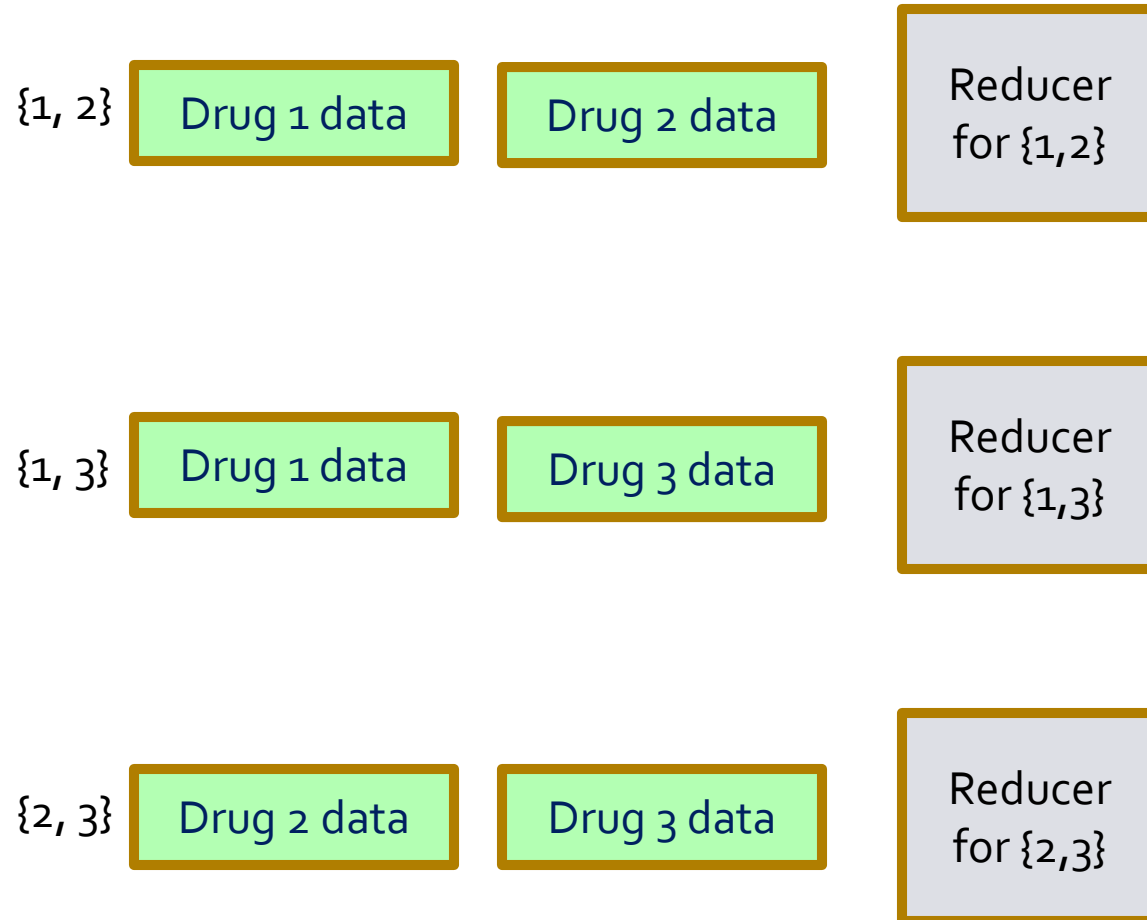
Example: Three Drugs



Example: Three Drugs



Example: Three Drugs



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.

The Improved Algorithm

- The team grouped 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.

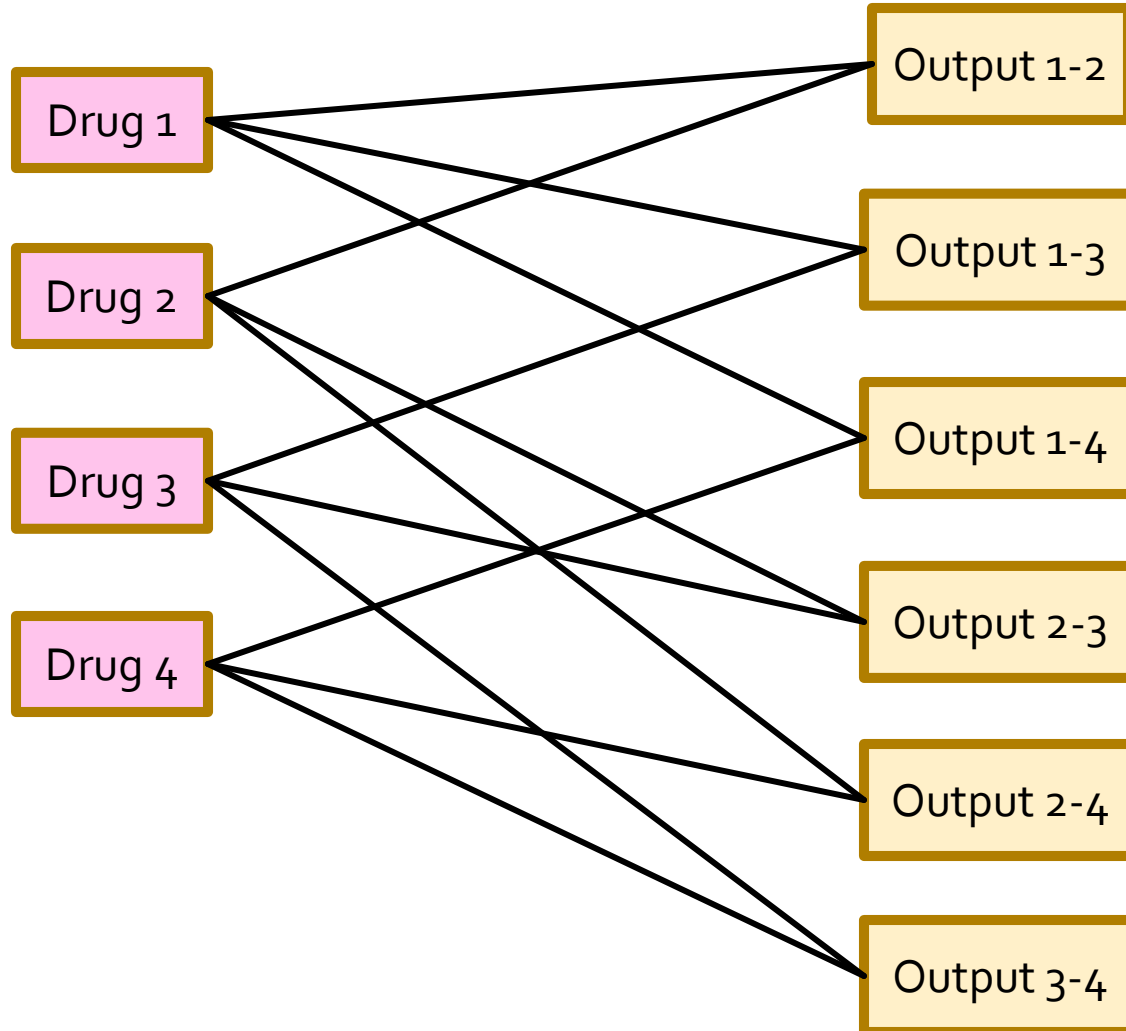
Outline of the Theory

Work of: Foto Afrati, Anish Das Sarma,
Semih Salihoglu, U
Reducer Size
Replication Rate
Mapping Schemas

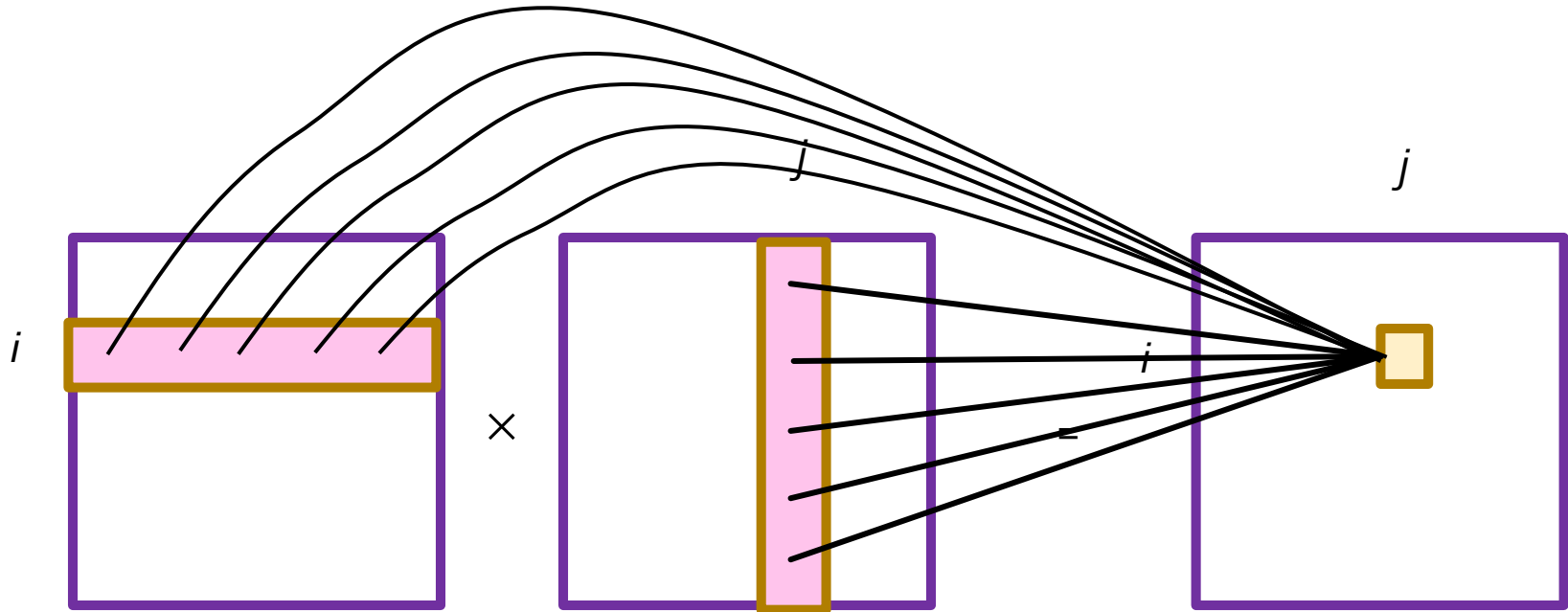
A Model for Map-Reduce Problems

1. A set of *inputs*.
 - *Example*: the drug records.
2. A set of *outputs*.
 - *Example*: one output for each pair of drugs, telling whether a statistically significant interaction was detected.
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



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.

Replication Rate

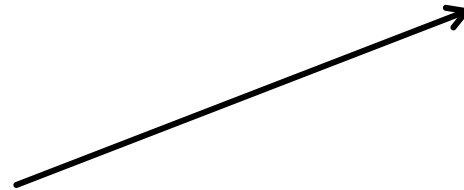
- The average number of key-value pairs created by each mapper is the *replication rate*.
 - Denoted r .
- Represents the communication cost per input.

Example: Drug Interaction

- Suppose we use g groups and d drugs.
- A reducer needs two groups, so $q = 2d/g$.
- Each of the d inputs is sent to $g-1$ reducers, or approximately $r = g$.
- Replace g by r in $q = 2d/g$ to get $r = 2d/q$.

Tradeoff!

The bigger the reducers,
the less communication.



Upper and Lower Bounds on r

- What we did gives an upper bound on r as a function of q .
- A solid investigation of MapReduce algorithms for a problem includes lower bounds.
 - Proofs that you cannot have lower r for a given q .

Proofs Need Mapping Schemas

- A *mapping schema* for a problem and a reducer size q is an abstraction of a MR algorithm.
- It assigns 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.
 - If some output is not covered, we can't compute that output.

Mapping Schemas – (2)

- Every MapReduce algorithm has a mapping schema.
- The requirement that there be a mapping schema is what distinguishes MapReduce algorithms from general parallel algorithms.

Example: Drug Interactions

- d drugs, reducer size q .
- Each drug has to meet each of the $d-1$ other drugs at some reducer.
- If a drug is sent to a reducer, then at most $q-1$ other drugs are there.
- Thus, each drug is sent to at least $\lceil (d-1)/(q-1) \rceil$ reducers, and $r \geq \lceil (d-1)/(q-1) \rceil$.
 - Or approximately $r \geq \lceil d/q \rceil$.
- Half the r from the algorithm we described.
- Better algorithm gives $r \leq d/q + 1$, so lower bound is actually tight.

The Hamming-Distance = 1 Problem

The Exact Lower Bound
Matching Algorithms

Definition of HD₁ Problem

- 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:** $r \geq b/\log_2 q$.
 - (Part of) the proof later.

Inputs Aren't Really All There

- If all bit strings of length b are in the input, then we already **know** the answer, and running MapReduce is a waste of time.
- A more realistic scenario is that we are doing a similarity search, where some of the possible bit strings are present and others not.
 - **Example**: Find viewers who like the same set of movies except for one.
- We can adjust q to be the **expected** number of inputs at a reducer, rather than the maximum number.

Algorithm With $q=2$

- We can use one reducer for every output.
- Each input is sent to b reducers (so $r = b$).
- Each reducer outputs its pair if both its inputs are present, otherwise, nothing.
- **Subtle point:** if neither input for a reducer is present, then the reducer doesn't really exist.

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 and outputs pairs at distance 1.

Splitting Algorithm

- Assume b is even.
- Two reducers for each string of length $b/2$.
 - Call them the *left* and *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, $r = 2$; $q = 2^{b/2}$.

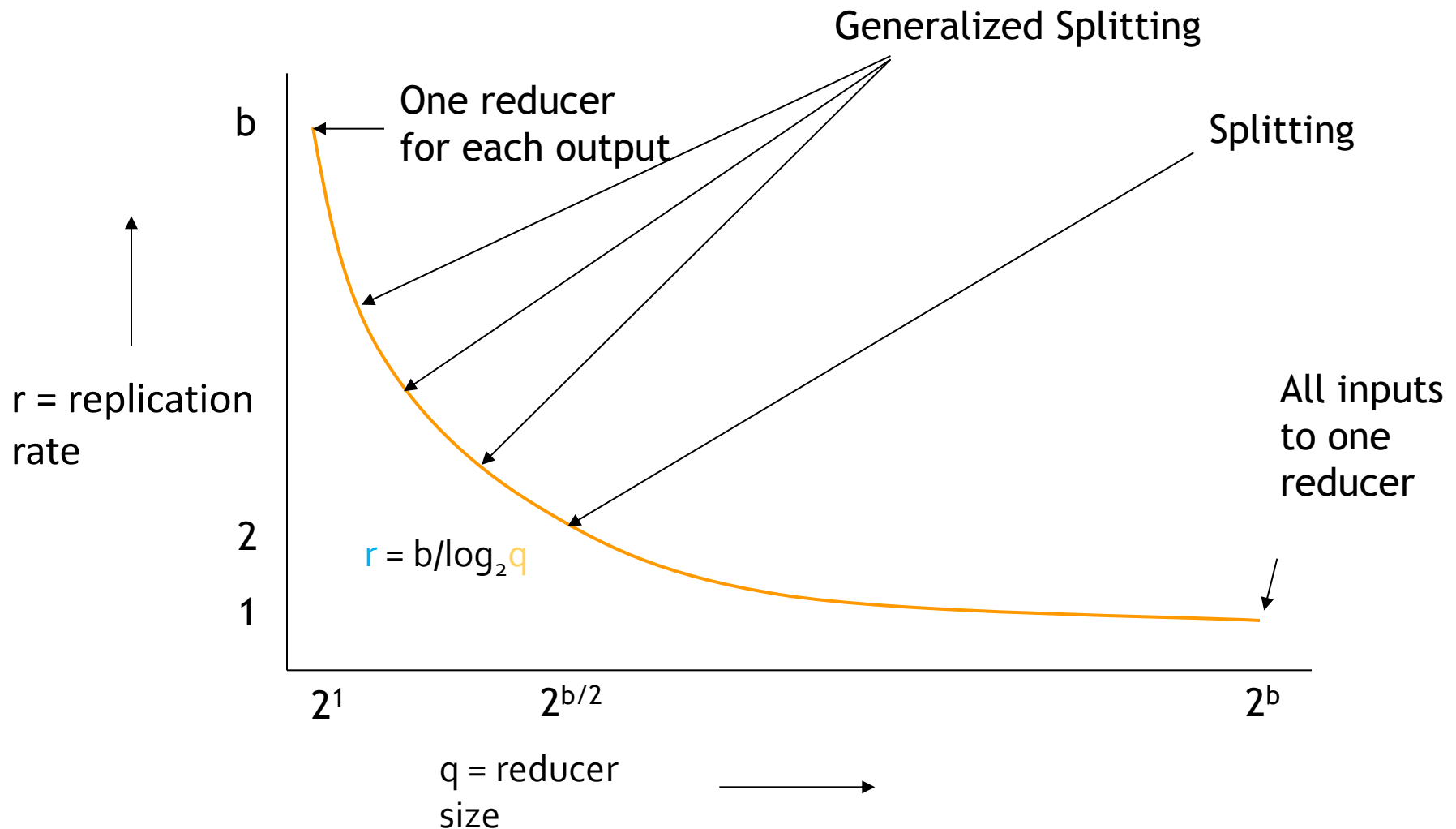
Recipe for Lower Bounds on r

1. Get an *upper* bound on the number of outputs that a reducer of size q can cover.
 2. Divide the number of outputs by (1) to get a *lower* bound on the number of reducers needed.
 3. Multiply q by (2) to get a *lower* bound on the total communication.
 4. Divide (3) by the number of inputs to get a *lower* bound on r .
- **Important point:** if the function from (1) grows at least linearly (it always does), then there is no need to consider more-but-smaller reducers.

Proof That $r \geq b/\log_2 q$

1. **Lemma:** A reducer of size q cannot cover more than $(q/2)\log_2 q$ outputs.
 - Induction on b ; proof omitted.
2. $(b/2)2^b$ outputs must be covered, so there are at least $p = (b/2)2^b / ((q/2)\log_2 q) = (b/q)2^b / \log_2 q$ reducers.
3. Sum of inputs over all reducers $\geq pq = b2^b / \log_2 q$.
4. Replication rate $r = pq/2^b = b/\log_2 q$.

Algorithms Matching Lower Bound



Matrix Multiplication

Lower Bound for One-Round Algorithms
An Optimal One-Round Algorithm
But Two Rounds Are Better Than One

Summary of MM Results

- **Problem:** multiply two n -by- n matrices.
- A reducer of size q produces $\leq q^2/4n^2$ outputs.
- Thus at least $4n^4/q^2$ reducers and $4n^4/q$ total communication.
- Implies $r \geq 2n^2/q$ replication rate.
- A generalization of the one-round algorithm given in MMDS matches the lower bound.
- But there is a two-round MM algorithm that uses total communication $4n^3/\sqrt{q}$.

Research Questions

- Get matching upper and lower bounds for the Hamming-distance problem for distances greater than 1.
- **Ugly fact:** For $HD=1$, you cannot have a large reducer with all pairs at distance 1; for $HD=2$, it is possible.
 - Consider all inputs of weight 1 and length b .

Research Questions – (2)

- It appears very hard to extend this theory to more than one MapReduce round.
 - Exception is the “two rounds better than one” result for Matrix Multiplication.
 - Are there other problems for which you can show “two better than one”?
 - Can you get a lower bound on total communication cost for multiround solutions to some problem?

Research Questions – (3)

- What happens if inputs are of different sizes?
 - **Example:** the drug records were only 1MB on average; some were actually much longer than others.
 - Put a limit not on the number of inputs at a reducer but on the total size of the inputs.