MapReduce Algorithms

Quick Introduction to MapReduce
An Example of the Design Problem
Reducer Size Vs. Replication Rate
Some Optimal Algorithms: Hamming distance, matrix multiplication, marginals

Jeffrey D. Ullman
Stanford University



Contributors

- Foto Afrati, National Technical Univ. of Athens
- Anish Das Sarma, Google/Troo.ly
- Semih Salihoglu, Stanford/Waterloo
- Shantanu Sharma, Ben Gurion Univ./UC Irvine
- Jonathan Ullman, Northeastern Univ.

Overview

- MapReduce and similar systems offer benefits at a cost.
  - Easy parallelism without writing parallel code.
  - Automatic management of failures.
- One cost is a subtle limitation on the algorithms that are implemented efficiently using these systems.
- This talk is about the nature of those limitations and some of the algorithms that are optimal under these constraints.

Free Stuff

- Free download of text *Mining of Massive Datasets* by Jure Leskovec, Anand Rajaraman, and U at www.mmds.org
- MOOC planned for Stanford-On-Line starting Oct. 11, 2016. Signup at mmds.lagunita.stanford.edu

MapReduce

Mappers and Reducers
Associated Costs

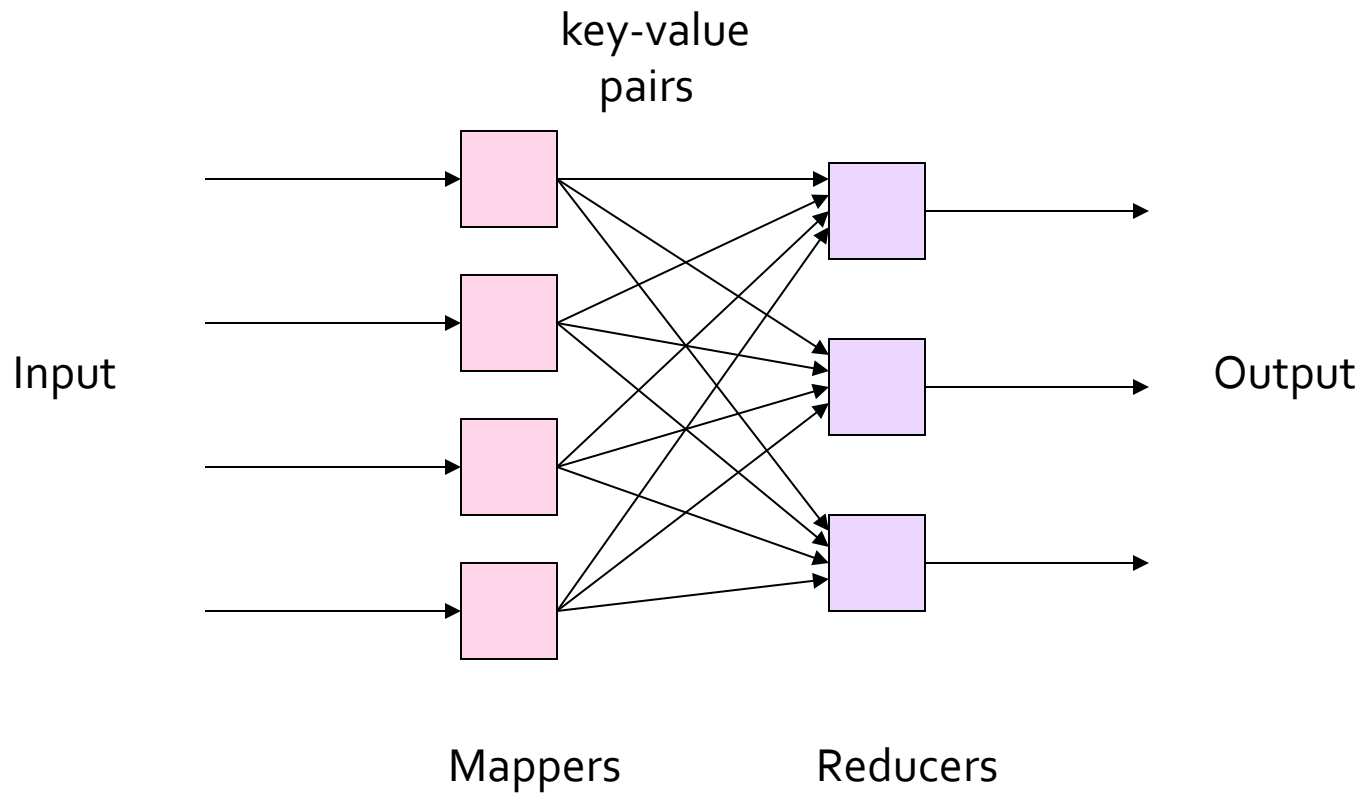
MapReduce in a Nutshell

- A MapReduce job starts with a collection of inputs of a single type.
- Apply a user-written *Map function* to each input, in parallel.
  - *Mapper* = application of the Map function to a single input.
    - Usually many mappers are grouped into a *Map Task*.
- The output of the Map function is a set of 0, 1, or more *key-value pairs*.
- The system sorts all the key-value pairs by key, forming key-(list of values) pairs.

In a Nutshell – (2)

- Another user-written function, the *Reduce function*, is applied to each key-(list of values).
  - Application of the Reduce function to one key and its list of values is a *reducer*.
    - Often, many reducers are grouped into a *Reduce Task*.
- Each reducer produces some output, and the output of the entire job is the union of what is produced by each reducer.

MapReduce Pattern



Cost of a MapReduce Algorithm

1. Execution time of the mappers and reducers.
2. Communication cost of transmitting the output of the mappers to the location of the proper reducer.
   - Usually, many compute nodes execute tasks in parallel, so there is little chance that the source and destination of a key-value pair are the same.
   - Often, communication cost dominates.

The All-Pairs Problem

Motivation: Drug Interactions
A Failed Attempt
Lowering the Communication

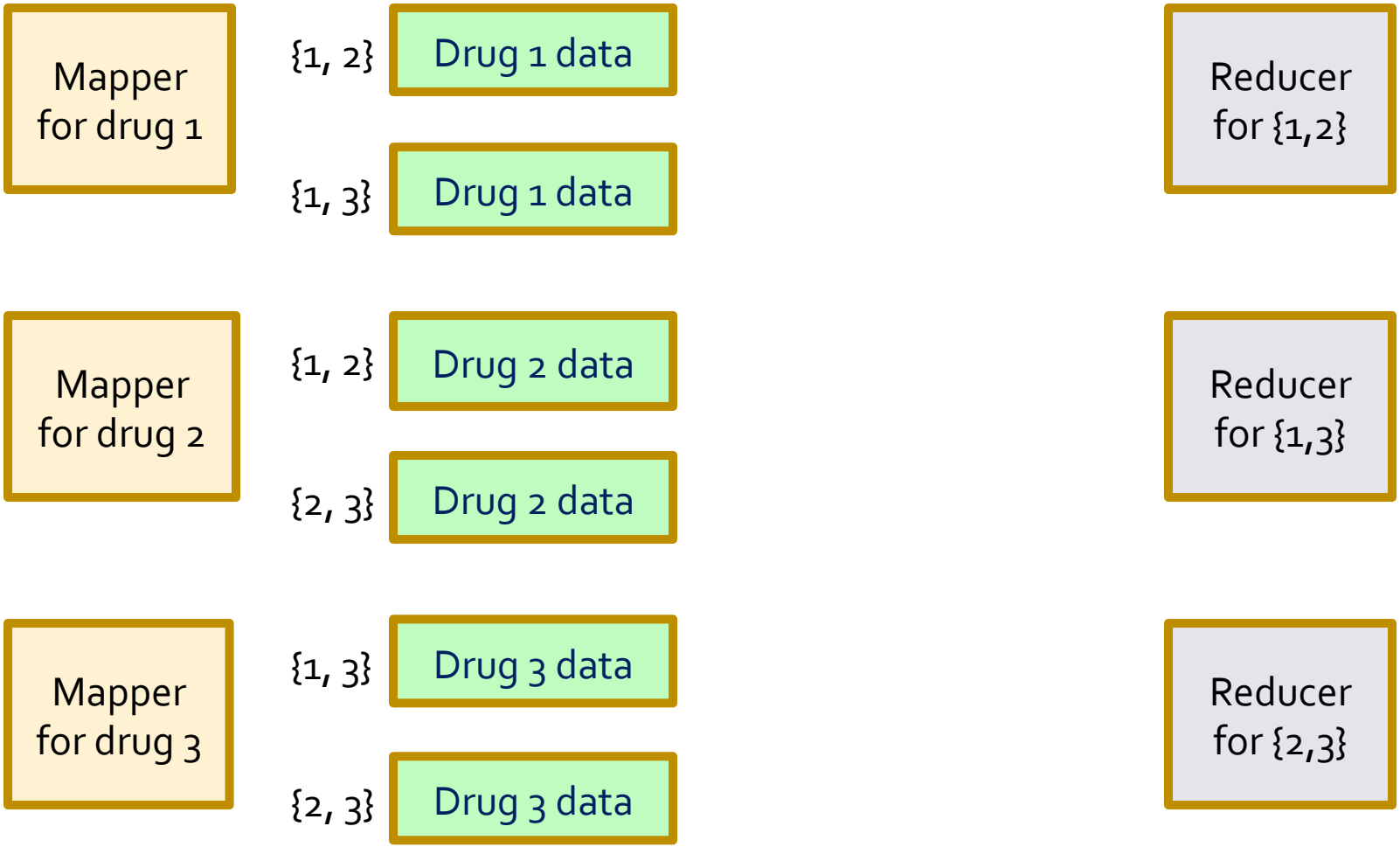
The Drug-Interaction Problem

- A real story from Stanford's CS341 data-mining project class.
- 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.

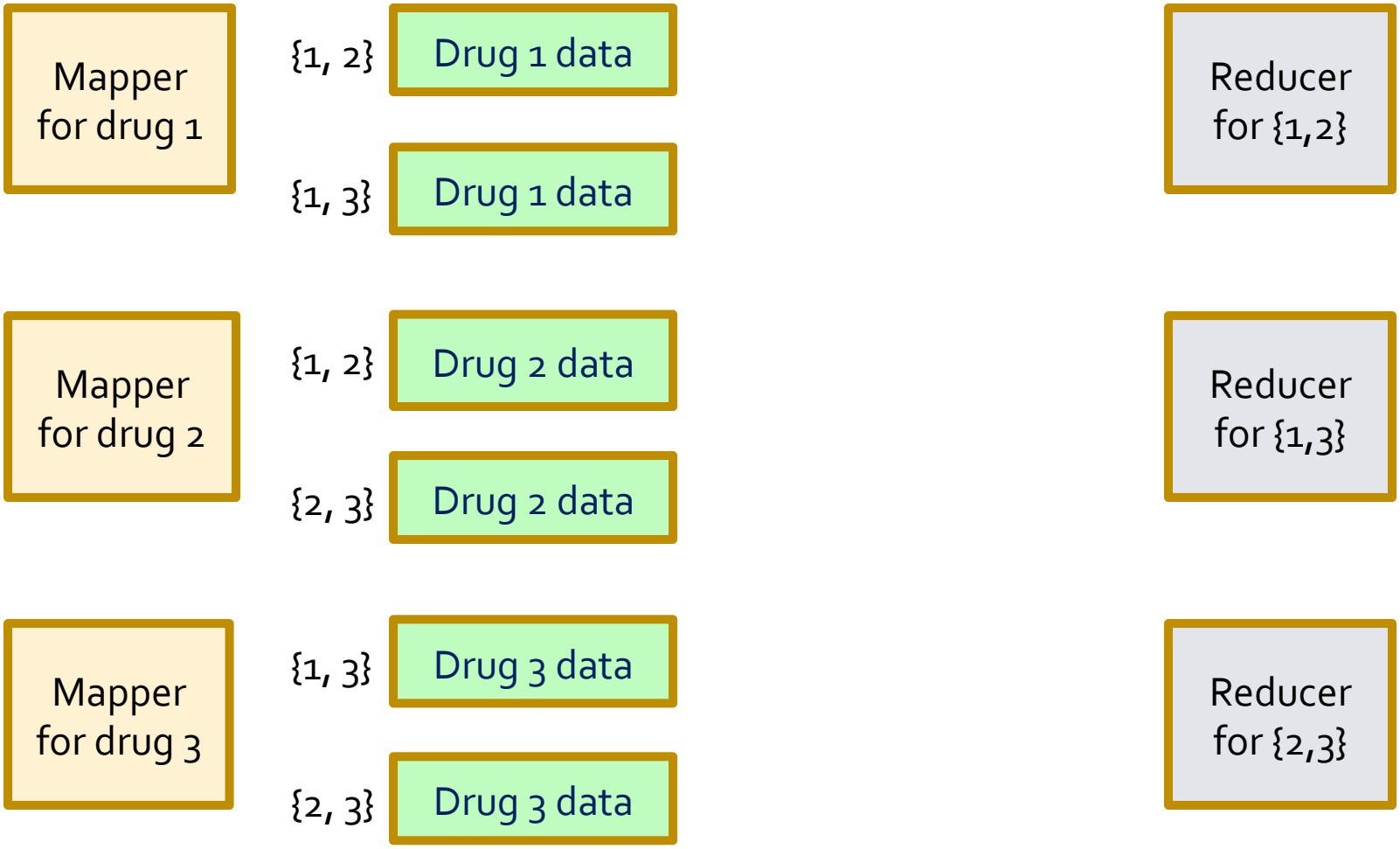
Initial Map-Reduce Algorithm

- 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: $(\{i, j\}, [R_i, R_j])$.

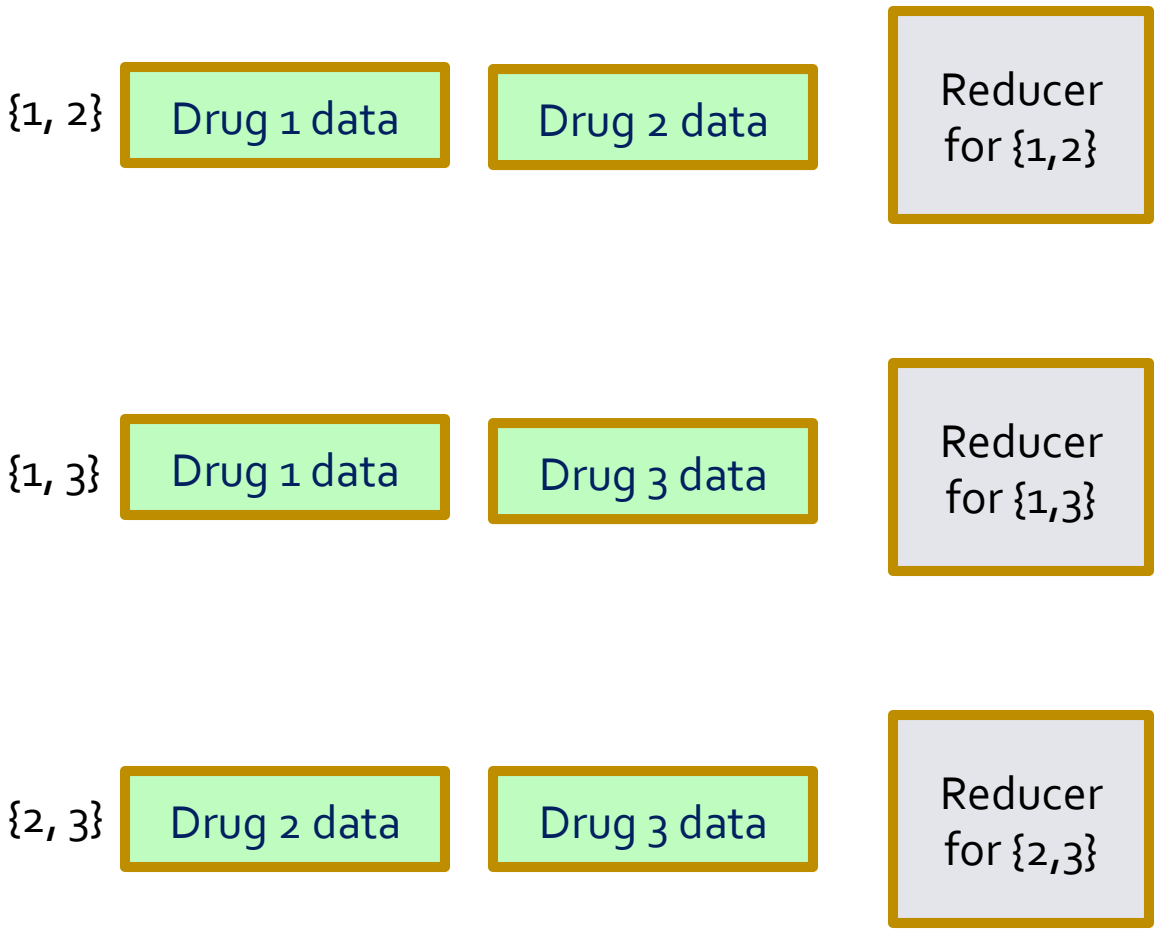
Example: Three Drugs



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.
- Keys are pairs of group numbers.
- Mapper for drug *i* produces 29 key-value pairs.
  - Each key is the set containing i's group 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*.

Improved Algorithm – Continued

- 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.

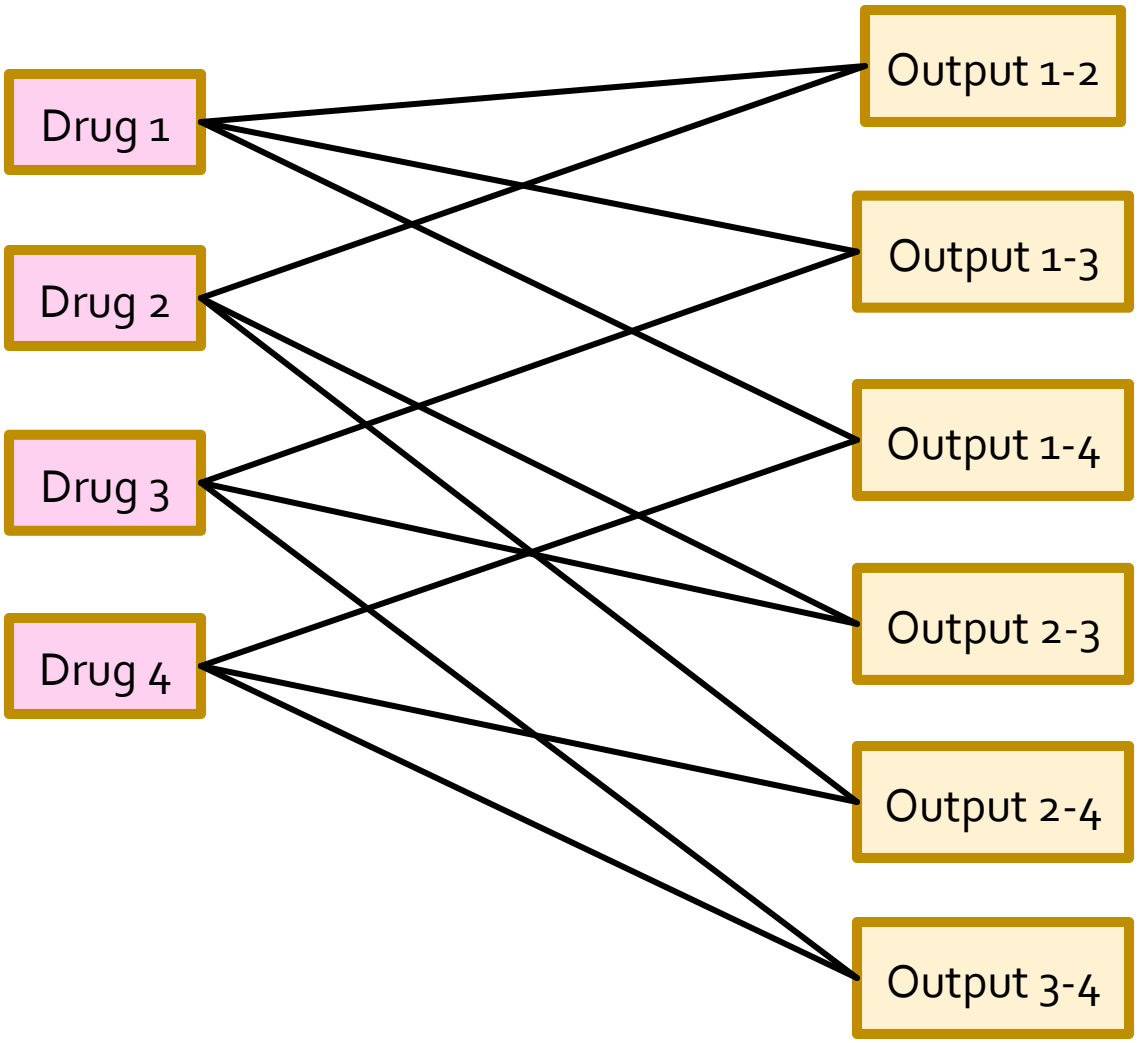
Theory of Map-Reduce Algorithms

- Reducer Size
- Replication Rate
- Mapping Schemas
- Lower Bounds

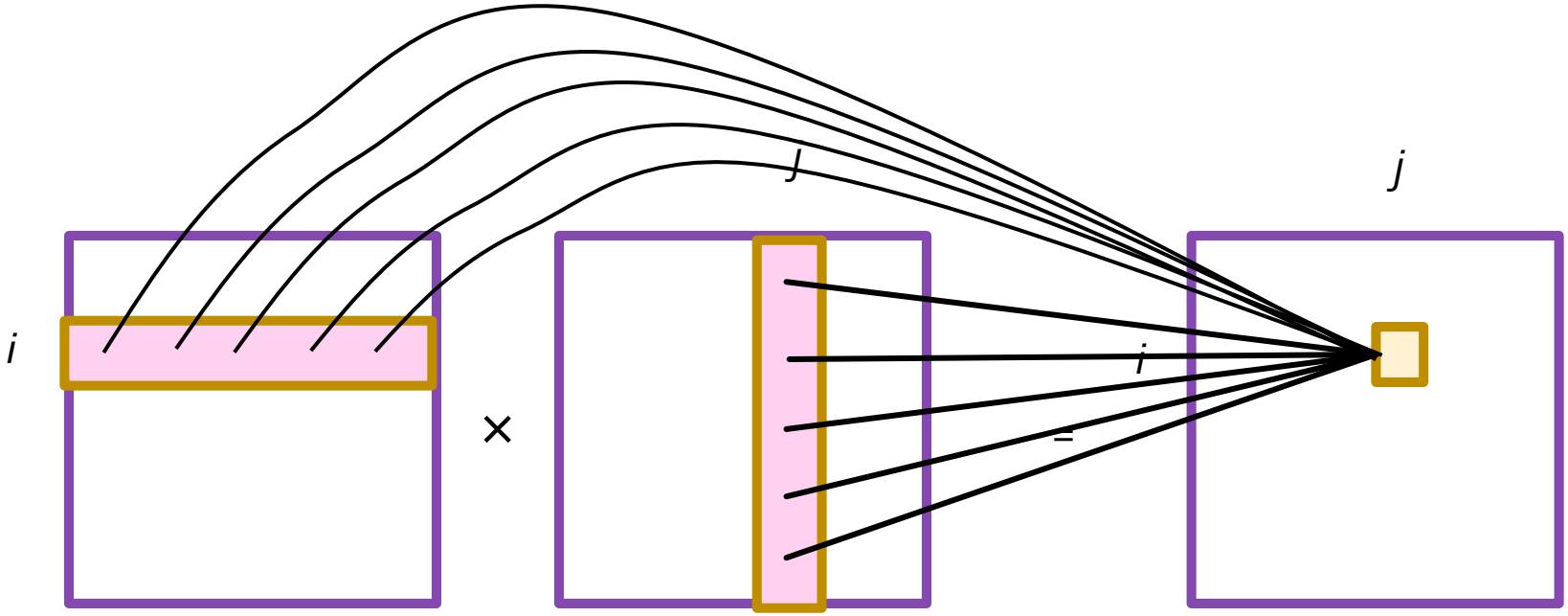
A Model for MapReduce Algorithms

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

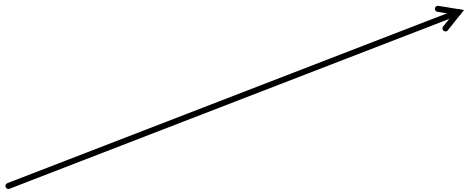
- *Reducer size*, denoted q, is the maximum number of inputs that a given reducer can have.
  - I.e., the length of the value list.
- Limit might be based on how many inputs can be handled in main memory.
- Or: make q low to force lots of parallelism.

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 there are 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.

Lower Bound on r

- What we did gives an upper bound on r as a function of q.
- We also need a lower bound on r for a given q.
- In other words, is the family of algorithms just described the best that can be done using MapReduce?

Proofs Need Mapping Schemas

- 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.
     - 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 $r \geq d/q$.
- Half the r from the algorithm we described.
- Better algorithm gives $r = d/q + 1$, so lower bound is actually tight.

The HD1 Problem

Some Optimal Algorithms
Matching Lower Bound

Definition of HD1 Problem

- Given a set of bit strings of length b, find all those that differ in exactly one bit.
  - I.e., their *Hamming distance* is 1.
- 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 = $b/\log_2 q$.

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.

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.
- Input 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

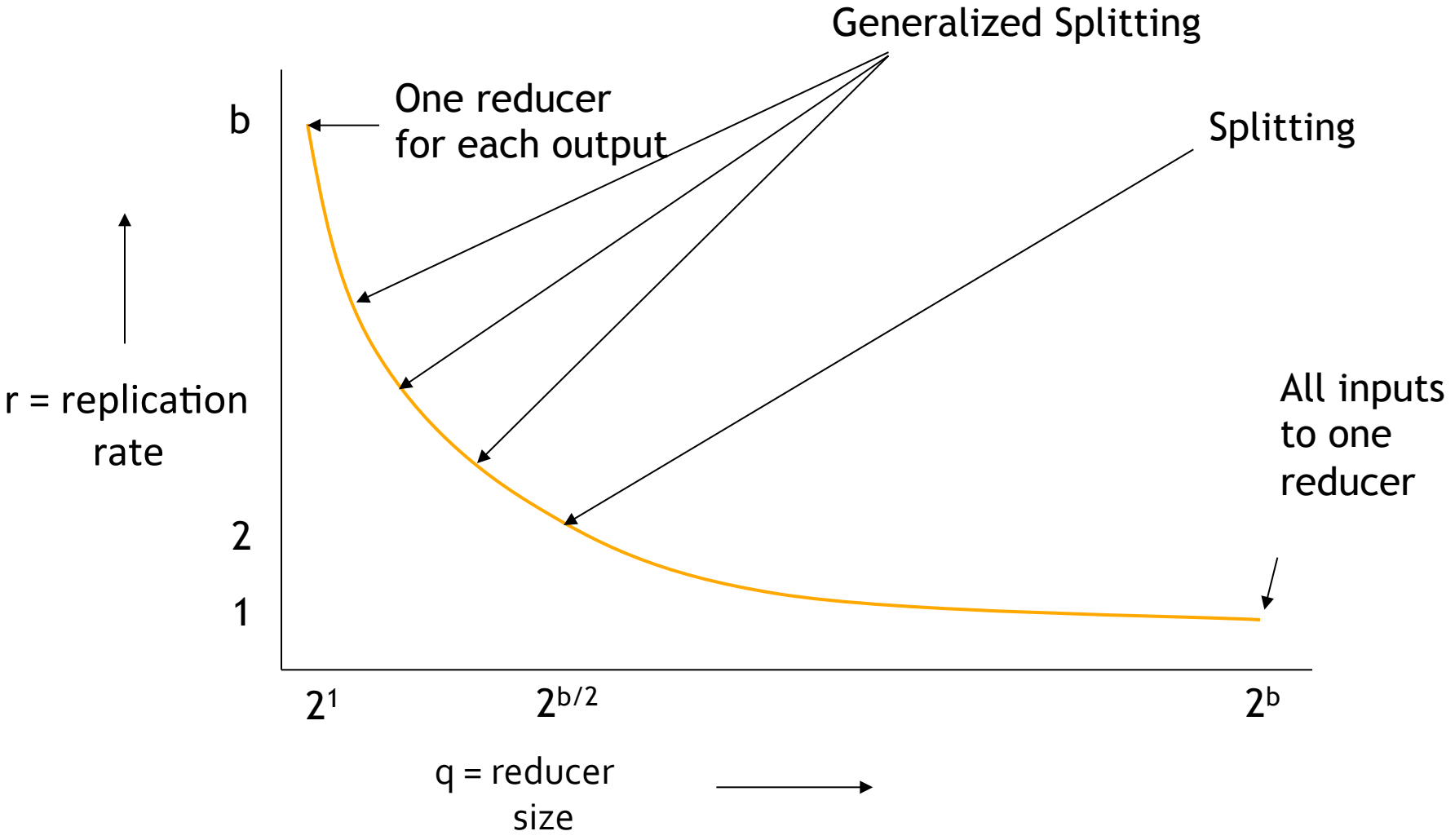
1. Get an *upper* bound on the number of outputs that a reducer of size q can cover.
2. Divide the number of outputs by (1) to get a *lower* bound on the number of reducers needed.
3. Multiply q by (2) to get a *lower* bound on the total communication.
4. Divide (3) by the number of inputs to get a *lower* 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



Open Problem

- What can you say about the Hamming-distance problem for distances greater than 1?

Matrix Multiplication

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

Summary of MM Results

- Multiply two n-by-n matrices.
- A reducer of size q can produce at most $q^2/4n^2$ outputs.
- Implies $r \geq 2n^2/q$ replication rate and lower bound $4n^4/q$ on total communication.
- The obvious algorithm matches the lower bound.
- But there is a two-round MM algorithm that uses total communication $4n^3/\sqrt{q}$.

Open Problems

1. Are there other problems for which you can assert two rounds of MapReduce are better than one?
2. Can you say anything about lower bounds on communication for multiround algorithms?

Marginals

Data Cubes
Simplifying Assumptions
Covering Numbers
Isoperimetric Inequality for Marginals

Data Cubes

- Many kinds of data can be organized as a "cube," where the sides of the cube are certain *dimension* attributes of a relation.
  - Examples: Items, Purchasers, Stores, Dates, …
- Points of the cube represent *facts*, e.g., a certain item purchased by a certain buyer at a certain store on a certain date.
- Other *dependent* attributes represent aggregatable quantities about a fact.
  - Examples: Price or Quantity.

Marginals

- Useful queries often involve summing dependent attributes over some but not all of the dimension attributes.
  - Example: For each Store and Date, find the total sales of all Items to all Purchasers.
- A *marginal* is the sum of a dependent attribute over all points in the data cube that have fixed values for certain dimension attributes.
- Say a marginal is *k-th order* if it sums over k dimension attributes (and fixes the other dimension attributes).

A Simplifying Assumption

- Let our data cube have n dimensions.
- For each dimension, assume there are exactly d distinct values.
- Makes the discussion of algorithms for computing marginals simpler, but its violation doesn't change results that much.
  - Because costs vary logarithmically with d.

Using MapReduce for Marginals

- Goal: compute all k-th order marginals using one round of MapReduce. Strategy:
  1. Pick a reducer size $q = d^m$ for some $m \geq k$.
  2. Pick some sets S of m dimensions.
  3. For each S, use a *family* of reducers, consisting of one reducer for each of the d^{n-m} assignments of values to the dimensions not in S.
  4. Send each point of the data cube to that member of the family for which it agrees in the n-m fixed dimensions.
     - So replication rate = # of sets S chosen.

Intuition

- We're pretending that we want to compute an m-th order marginal at each reducer in the family.
- But we'll really get many k-th order marginals from one reducer.
- This approach turns out to be the best way to use reducers of size d^m.

Picking Sets of m Attributes

- The reducer family for set of m dimensions S can compute all the marginals that aggregate over a set T of size k iff $T \subseteq S$.
  - Why? In each reducer in the family, fix the m-k dimensions in S-T in all possible ways.
    - Thus, among all the reducers of the family, all dimensions outside T are set in all possible ways.

But Which Sets Should We Use?

- If we need to compute all k-th order marginals, we need to know that for every set T of k dimensions, there is some selected set S of m dimensions such that $T \subseteq S$.
- The fewer sets S we choose, the lower the replication rate.

Example: Picking Sets

- Dimensions = ABCDE, k=2, m=3.
- We can pick ABC, ADE, BDE, CDE.
- Covers all 10 pairs.
- Best possible, because there are 10 pairs, and a triple can only cover three pairs.

Covering Numbers

- C(n,m,k) = the smallest number of sets of size m such that every set of k out of n elements is contained in one of these sets of size m.
- Lower Bound: $C(n,m,k) \geq \binom{n}{k} / \binom{m}{k}$
- Proof: This is the number of sets of size k divided by the number of sets of size k that one set of size m can contain.

Theorem: Upper Bound on Marginals Covered by a Reducer

- Consider an n-dimensional cube of side d.
- Consider a set S of q points within that cube.
- Then the greatest number of subcubes of dimension k that can be contained within S is $(q/d^k)\binom{\log_d q}{k}$
- If $q = d^m$, then this upper bound is $d^{m-k}\binom{m}{k}$
- That is exactly the number of subcubes of dimension k that we cover if we give a reducer all points that share fixed values in n-m dimensions and vary in the other m dimensions.

Open Problem

- How does this bound generalize if the sides of the cube are not a constant d, but vary?

Some Extensions

Dimensions with Different Extents

Building k-th Order Marginals from Marginals of Smaller Order

What if Dimension Extents Vary?

- We do not really expect dimensions to have the same extent d.
- We can still use a lot of the "covering number" ideas to get reasonable ways to partition the problem among (families of) reducers.
- Intuition: "weighed" covering, where each dimension has a weight (logarithm of the extent), and sets can vary in size but with a limit on total weight.

Open Problem

- What can you say about the “weighted covering problem,” where covering sets S can be of different sizes, as long as the sum of the weights of their elements is below some bound?

Large Marginals from Smaller

- We often want to compute the marginals of a data cube starting from marginals of a small order, say 1, and building up to higher-order marginals in steps.
- Yields a new combinatorial problem similar to "covering numbers."

The New Covering Problem

- *Cover* sets of k dimensions with pairs of sets (S,T), where S is of size m and T of size j.
  - Covers all sets U such that $T \subseteq U \subseteq (S \cup T)$.
- Intuition: One reducer (of size $q = d^m$) gets all the j-th order marginals that aggregate over the j dimensions in T and have fixed values for all dimensions except those in S and T.
- Define *D(n,m,j,k)* = the number of needed pairs (S,T), where |S| = m, |T| = j, and every set of size k out of the n "dimensions" is covered by some pair.

Lower Bound on D

- $D(n,m,j,k) \geq \binom{n}{k} / \binom{m}{k-j}$
- One interesting result: lower bound $n(n-1)/4m$ is (almost) attainable for $D(n,m,1,2)$.
  - I.e., computing 2^{nd}-order marginals from 1^{st}-order marginals, using reducers of size $q = d^m$.
- Also: $D(n,m,j,k) \leq \Sigma_{j<i<n-k+j} \binom{i-1}{j-1} C(n-i,m,k-j)$.

Open Problems

1. There remain lots of open questions regarding the values of C(n,m,k).
   - Especially useful are recursive constructions, because these make it easier to design algorithms for the realistic case where extents vary among dimensions.
2. Almost nothing is known about the values of D(n,m,j,k).