Massive Joins in Parallel

The Relational Join
Parallel Hash Join Algorithm
MapReduce
Multiway Joins
Estimating the Size of a Union of Sets

Jeffrey D. Ullman
Stanford University/Infolab

Free Book

- Mining of Massive Datasets by Jure Leskovec, Anand Rajaraman and Jeff Ullman.
- Published by Cambridge University Press, but available for free download at www.mmds.org

The Relational Join Operation

- *Relations* are tables with names (*attributes*) for the columns.
- You *join* two or more relations by sticking together tuples from each relation that agree on the shared attributes.

Example: Join

City	Country
Tokyo	Japan
Kyoto	Japan
Istanbul	Turkey
Ankara	Turkey

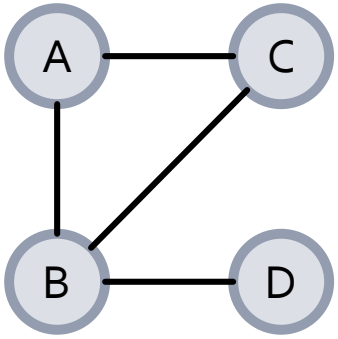
Country	Continent
Japan	Asia
Turkey	Asia
Turkey	Europe

City	Country	Continent
Tokyo	Japan	Asia
Kyoto	Japan	Asia
Istanbul	Turkey	Asia
Ankara	Turkey	Asia
Istanbul	Turkey	Europe
Ankara	Turkey	Europe

The Facebook Friends Relation

- There are about 2 billion Facebook users.
- On average, each has about 300 friends.
- We can represent the “friends” relation as an undirected graph.
- Equivalently, we can represent “friends” as a relation with two columns Person1 and Person2.
  - Rows are pairs of people who are Facebook friends.

Example: Facebook Friends



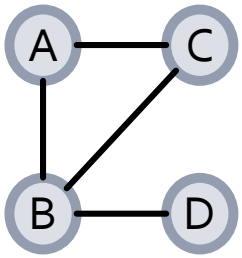
Person1	Person2
A	B
A	C
B	A
B	C
B	D
C	A
C	B
D	B

In reality, the Facebook friends graph has about $2*10^9$ nodes and $3*10^{11}$ edges. The relation has about $6*10^{11}$ rows.

Friends of Friends

- Find paths of length 2 in the graph, or
- Join the Friends relation with itself.
  - But you need to rename the columns to make it work.

Example: Friends of Friends



Person1	Person2
A	B
A	C
B	A
B	C
B	D
C	A
C	B
D	B

Person2	Person3
A	B
A	C
B	A
B	C
B	D
C	A
C	B
D	B

Notice some result pairs appear more than once, e.g., (A,A).

Result is first and third columns, e.g., (A, C).

Person1	Person2	Person3
A	B	A
A	B	C
A	B	D
. . .	. . .	. . .

15 more rows

How Big is Friends of Friends?

- If the Friends relation has $6*10^{11}$ rows, and the average person has 300 friends, then the join of Friends with itself has $1.8*10^{14}$ rows.
- Worse: to compute the join naively, you need to compare each row of one relation with each row of the second relation, to see if their common attribute agrees.
  - Takes $(6*10^{11})^2 = 3.6*10^{23}$ comparisons.
- You can't compute that on one processor; you need parallelism and a clever way to find rows that match.

Parallel Hash Join

- To join relations with columns (X, Y) and (Y, Z), hash each row to a bucket based only on the Y value.
- Each bucket is assigned one processor.
  - You can have as many buckets as there are processors.
- Any pair of rows that join will be assigned to the same bucket/processor, so the union of joinable rows each processor finds is the complete answer.

Example: Parallel Hash Join

- Suppose we use 2 processors on our little "friends" example.
- Suppose A and B hash to one bucket and C and D to the other bucket.

Processor/bucket 1

Person1	Person2
A	B
B	A
C	A
C	B
D	B

Person2	Person3
A	B
A	C
B	A
B	C
B	D

Person1	Person2	Person3
A	B	A
A	B	C
A	B	D
. . .	. . .	. . .

Processor/bucket 2

Person1	Person2
A	C
B	C
B	D

Person2	Person3
C	A
C	B
D	B

Person1	Person2	Person3
A	C	A
A	C	B
B	C	A
B	C	B
B	D	B

MapReduce

- A programming system for getting programs to execute in parallel without having to think about parallelism.
- Two *stages*: Map and Reduce, with hidden work between the stages.
- The magic: you only have to write two small, serial programs – the *Map function* and the *Reduce function*; all the parallelism is done for you.

The Map Stage

- The MapReduce system (e.g., Hadoop or Spark) applies the Map function to each input, in parallel.
- Output of the Map function is a set of *key-value pairs*.

Example: Map function

- Suppose we join one copy of Friends, with columns (Person1, Person2) with another copy having columns (Person2, Person3).
- The Map function will take each row of either relation and produce a single key-value pair with:
  - Key = Person2 (regardless of which relation the row comes from).
  - Value = (i, x), where:
    - i = 1 or 2, depending on which relation the row is from.
    - x = value of Person1 (if i = 1) or Person3 (if i = 2).

Between the Stages

- The MapReduce system groups all the key-value pairs from all the applications of the Map function according to the key.
- For each key, a pair consisting of that key and the list of all the values associated with that key becomes one input for the Reduce stage.

Example: Between the Stages

Person*	Person*
A	B
A	C
B	A
B	C
B	D
C	A
C	B
D	B

- Consider our tiny "friends" example and the key A.
- The first copy of Friends has pairs (B,A) and (C,A) with Person2 = A.
- The second copy has pairs (A,B) and (A,C) with Person2 = A.
- So the system produces the pair

  (A, [(1,B), (1,C), (2,B), (2,C)]).

Reduce Stage

- A *Reduce function* is applied to each key-(list of values), and the output of the entire job is the union of the output produced by each application of the Reduce function.
  - Application of the Reduce function is called a *reducer*.
- Example: take each value of the form (1, x) on the list of values for key y, pair it with each value on the list of the form (2, z), and produce the output row (x, y, z).

Comments: Reducers

- Notice that all rows actually compared by a reducer really do join.
  - So this MapReduce algorithm looks at as few pairs of rows as possible.
- Usually, there are many more reducers than there are processors, so we group large numbers of reducers together into *Reduce tasks* and execute a Reduce task serially on a single processor.

A Harder Problem

- Suppose we want to find friends of friends of friends.
- Equivalent to finding nodes connected by paths of length 3 in the friends graph.
- Assuming 300 friends per person, the number of paths of length 3 is $2*10^9 \times 300^3 = 5.4*10^{16}$.
- Assuming people are represented by 32-bit integers (so rows are only 8 bytes), this result is over 400 petabytes in size.
  - Reduced considerably because there may be many paths connecting the same two people, but still huge.

Naïve Solution

1. Join Friends(Person1, Person2) with Friends(Person2, Person3).
2. Project out Person2 from the result and eliminate duplicates to get an intermediate relation with columns (Person1, Person3).
3. Join the intermediate relation with Friends(Person3, Person4).
4. Project out Person3 from this result and eliminate duplicates to get the answer, with columns (Person1, Person4).

Problems with Naïve Solution

- The result of Step 1 (the join of Friends with itself) has $1.8*10^{14}$ rows.
  - At 8 bytes per row = 1.5 petabytes.
- And the result of Step 3 (join of the intermediate relation with Friends) will be considerably larger.

A More Realistic Problem

- Change the question: Do we really need to know all pairs of people connected by paths of length 3?
  - Or would a count, for each person, of the number of their friends of friends of friends do?
  - That count would be enough to find *influencers* – people with a lot of people they can reach easily.

But We Need More …

- Even to count requires we compute the join of three relations and count rows in the result.
- So we are going to do two things:
  1. Make an approximate count of the number of friends of friends of friends for each person.
     - We can approximate the count using much less space than is needed to construct the join itself.
  2. Join three copies of Friends with one round of MapReduce.
     - Avoids constructing the intermediate join, which would otherwise have to be produced, even if we only want counts.

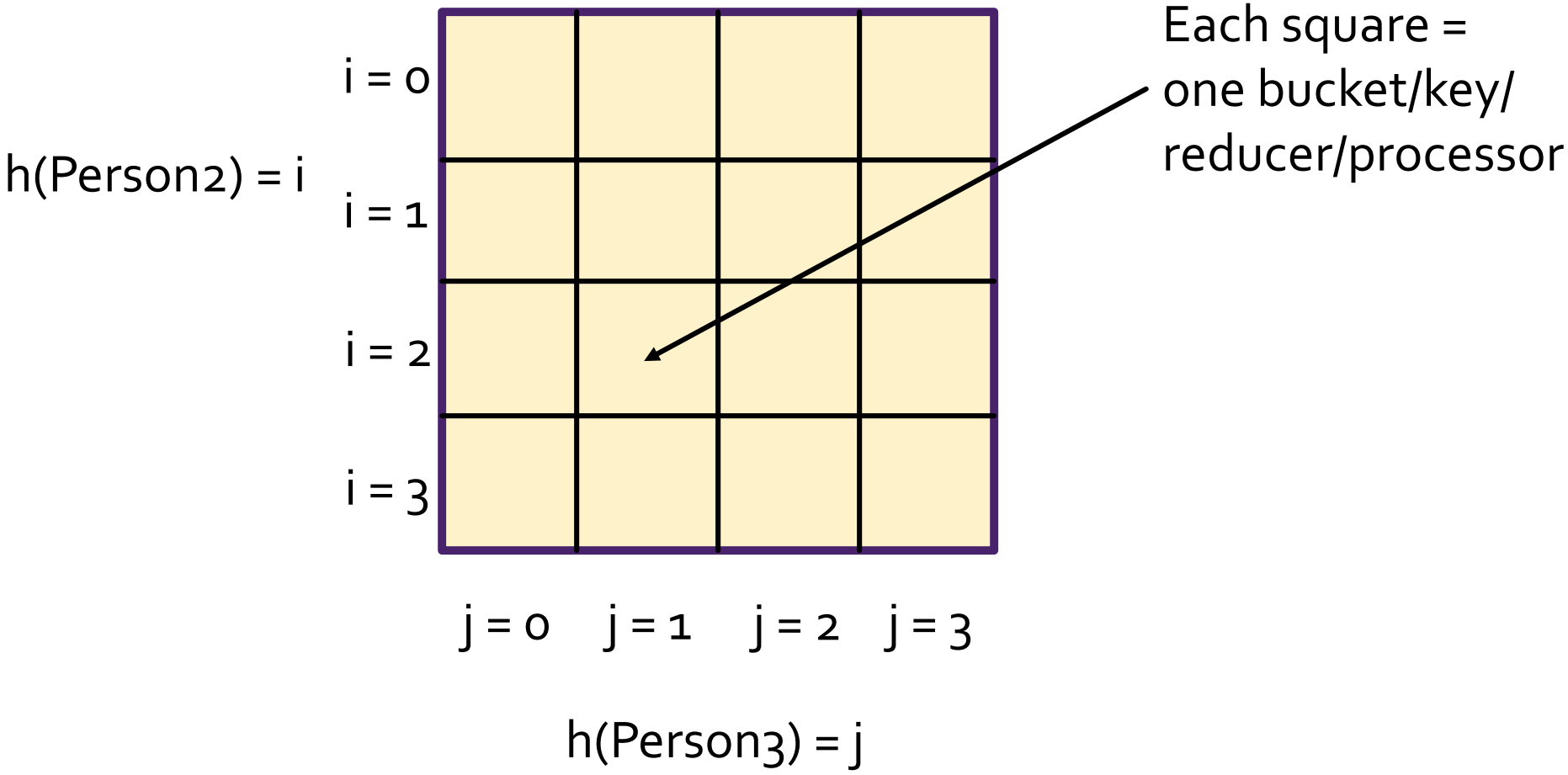
Three-Way Join of Friends

- We will join Friends(Person1, Person2), Friends(Person2, Person3), and Friends(Person3, Person4) by a single MapReduce job.
- Avoids constructing the intermediate relation.
- But we pay by sending each row to many reducers.
  - Increases the communication between the Map and Reduce stages.

Three-Way Join – (2)

- Pick a hash function h with a number of buckets, b.
  - We will hash Person2 and Person3 using h.
  - Pick b as large as possible, but you need b^2 processors working in parallel for this trick to work.
- Keys (reducers) are pairs of bucket numbers (i, j).

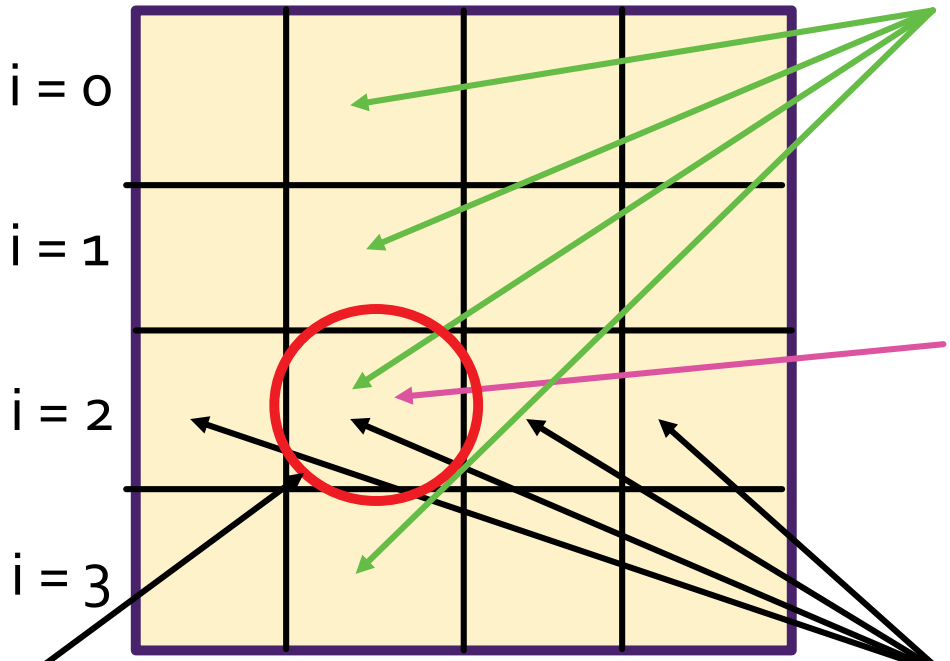
Example: b = 4


h(Person2) = i
i = 0
i = 1
i = 2
i = 3
Each square = one bucket/key/ reducer/processor
j = 0
j = 1
j = 2
j = 3
h(Person3) = j

Three-Way Join – Map Stage

- Take each row (p, q) from (Person1, Person2) and create b key-value pairs; the row is the value, and the keys have the form (h(q), x), for any of the b possible values of x.
- Take each row (p, q) from (Person2, Person3) and send it as a value associated only with key (h(p), h(q)).
- Take each row (p, q) from (Person3, Person4) and send that row as the value, with each key of the form (y, h(p)), for any of the b possible values of y.

Example: h(q) = 2; h(r) = 1



h(Person2) = i

h(Person3) = j

(r,s) from (Person3,Person4)

(q,r) from (Person2,Person3)

(p,q) from (Person1,Person2)

(p,q), (q,r), and (r,s) all meet here and only here.

How Much Communication?

- Suppose b = 10, so we use 100 processors.
- Each row of (Person1, Person2) and each row of (Person3, Person4) is sent to 10 processors or reducers.
- But each row of (Person2, Person3) is sent to only one processor/reducer.
- So the total number of rows communicated is $6*10^{11}$ (10 + 10 + 1) = $1.26*10^{13}$.
- Much smaller than the size of the join of two copies of Friends, which is $1.8*10^{14}$.
- And output is produced 100 times faster.

Two Pieces of Bad News

1. The more processors you use, the more communication.
   - At about 22,000 processors, there are as many rows communicated as there are rows in the join of Friends with itself.
2. The output size is still too large: $5.4*10^{16}$.

But Remember …

- We didn't ask for the entire join – just the number of friends of friends of friends for each person.
- But when we take the union of the outputs of the b^2 processors or reducers, there will be many duplicates.
  - So you can't just add the counts from each processor.
- We can't get the exact count without first seeing the 3-way join and examining the particular (Person1, Person4) pairs produced.

Flajolet-Martin Algorithm

- A way to estimate the size of a union of sets without computing each element of each set.
- Keep several (e.g. 100) *variables*, each of which is a small integer.
- For each variable V there is a hash function.
- Hash each input (user name, e.g.) to a bit string.
- The *tail length* of a bit string is the number of consecutive 0's at its end.
  - Example: the tail length of 010101 is 0, and the tail length of 101000 is 3.
- V records the largest tail length seen so far.

Flajolet-Martin Algorithm – (2)

- If the largest tail length is R, then variable V estimates you have seen 2^R different values.
  - That's roughly how many different values you need to get R 0's at end.
- Combine the estimates from different variables (a bit tricky) to get an accurate estimate of the number of different values seen.

Using Flajolet-Martin

- As each processor generates rows of the 3-way join, don't keep the row itself.
- Rather, the processor maintains, for each person p, the maximum tail length for each of the Flajolet-Martin variables.
  - Each variable only requires about 5 bits, since tail lengths bigger than 31 are unlikely.
- If (p, q, r, s) is generated, apply the hash function for each of the F-M variables V to s, and set V (for person p) to that tail length, if the tail is longer than the old value of V.

Using Flajolet-Martin – (2)

- 100 F-M variables require 500 bits, or 63 bytes.
- If a person is represented by 4 bytes, then each processor produces at most $2*10^9 \times (4 + 63) = 1.34*10^{11}$ bytes.
  - That is less than the size of the Friends relation.
- We need to estimate the size of the union of the outputs of all the processors.
- Trick: For each person and each F-M variable take the maximum of the tail lengths from each of the processors, and use those maximums to estimate the size of the union.

Points to Remember

1. Joins of relations are easy to do in parallel.
2. MapReduce is a great way to write parallel programs for many tasks, like the join, without having to "think parallel."
3. It sometimes makes sense to compute a join of more than 2 relations as one MapReduce job, especially if the join of any two of the relations is very large.
4. The Flajolet-Martin algorithm lets us estimate the size of the union of sets without having to compute the sets themselves.