CMPE 282 Cloud Services
MapReduce Design Patterns - Join

Instructor: Kong Li

Content

- What and Why
- MapReduce refresher
- Summarization Patterns
- Filtering Patterns
- Data Organization Patterns
- Join Patterns

Join Patterns

What: Bringing data sets together

Why: I want to mash my different data sources together

- **Reduce-side join**
- **Replicated join**
- Composite join
- **Cartesian product**

RDBMS Join Refresher

Table 5-1. Table A

User ID	Reputation	Location
3	3738	New York, NY
4	12946	New York, NY
5	17556	San Diego, CA
9	3443	Oakland, CA

Table 5-2. Table B

User ID	Post ID	Text
3	35314	Not sure why this is getting downvoted.
3	48002	Hehe, of course, it's all true!
5	44921	Please see my post below.
5	44920	Thank you very much for your reply.
8	48675	HTML is not a subset of XML!

Table 5-3. Inner Join of A + B on User ID

A.User ID	A.Reputation	A.Location	B.User ID	B.Post ID	B.Text
3	3738	New York, NY	3	35314	Not sure why this is getting downvoted.
3	3738	New York, NY	3	48002	Hehe, of course, it's all true!
5	17556	San Diego, CA	5	44921	Please see my post below.
5	17556	San Diego, CA	5	44920	Thank you very much for your reply.

Table 5-4. Left Outer Join of A + B on User ID

A.User ID	A.Reputation	A.Location	B.User ID	B.Post ID	B.Text
3	3738	New York, NY	3	35314	Not sure why this is getting downvoted.
3	3738	New York, NY	3	48002	Hehe, of course, it's all true!
4	12946	New York, NY	null	null	null
5	17556	San Diego, CA	5	44921	Please see my post below.
5	17556	San Diego, CA	5	44920	Thank you very much for your reply.
9	3443	Oakland, CA	null	null	null

Table 5-5. Right Outer Join of A + B on User ID

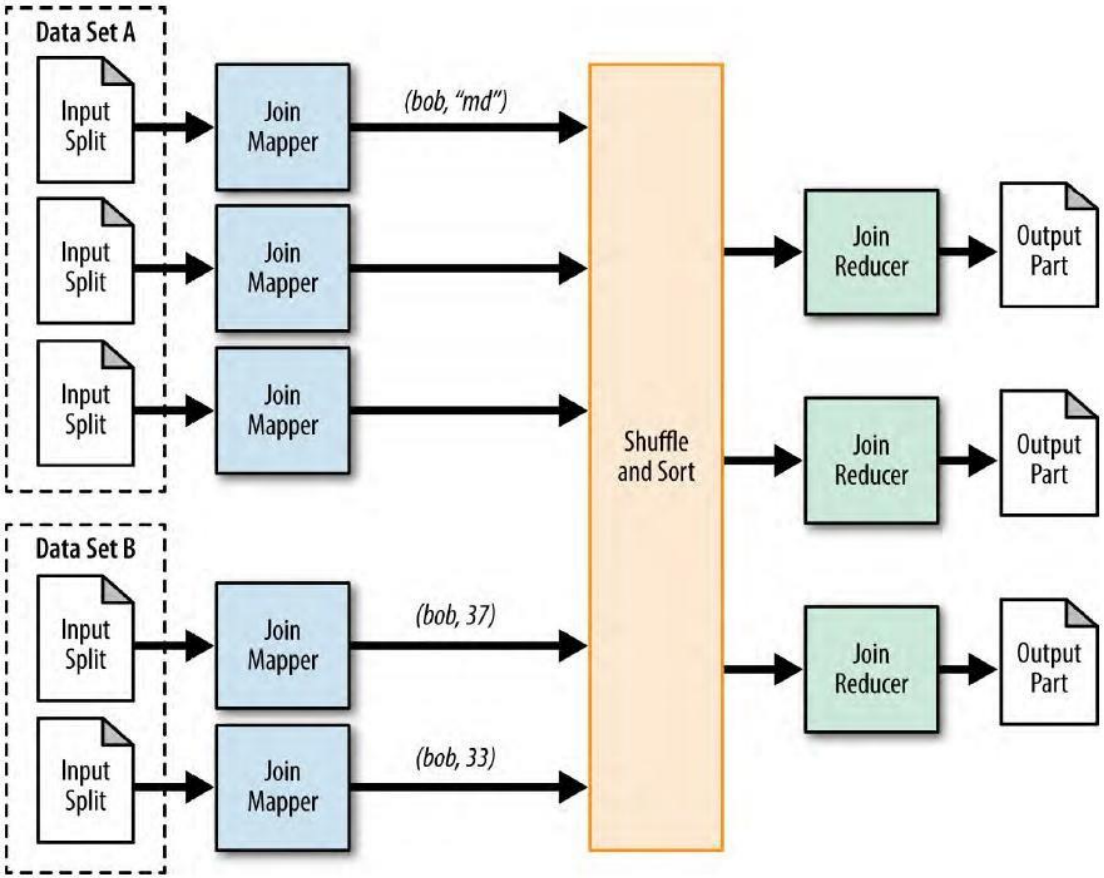
A.User ID	A.Reputation	A.Location	B.User ID	B.Post ID	B.Text
3	3738	New York, NY	3	35314	Not sure why this is getting downvoted.
3	3738	New York, NY	3	48002	Hehe, of course, it's all true!
5	17556	San Diego, CA	5	44921	Please see my post below.
5	17556	San Diego, CA	5	44920	Thank you very much for your reply.
null	null	null	8	48675	HTML is not a subset of XML!

Reduce Side Join 1/3

- Intent
  - Join large multiple data sets together by some foreign key
- Motivation
  - Simple to implement in Reducers
  - Supports all the different join operations (inner, outer)
  - No limitation on the size of your data sets
- Applicability
  - Multiple large data sets are being joined by a foreign key
  - You want the flexibility of being able to execute any join operation
  - A large amount of network bandwidth

Reduce Side Join 2/3

- Structure
  - Mapper: prepares [(foreign key, record)]
  - Reducer: receives (FK, list(values)) and performs join operation



Reduce Side Join 3/3

- SQL

  SELECT T1.ID, T1.A, T2.C

  FROM T1 [INNER|LEFT|RIGHT] JOIN T2

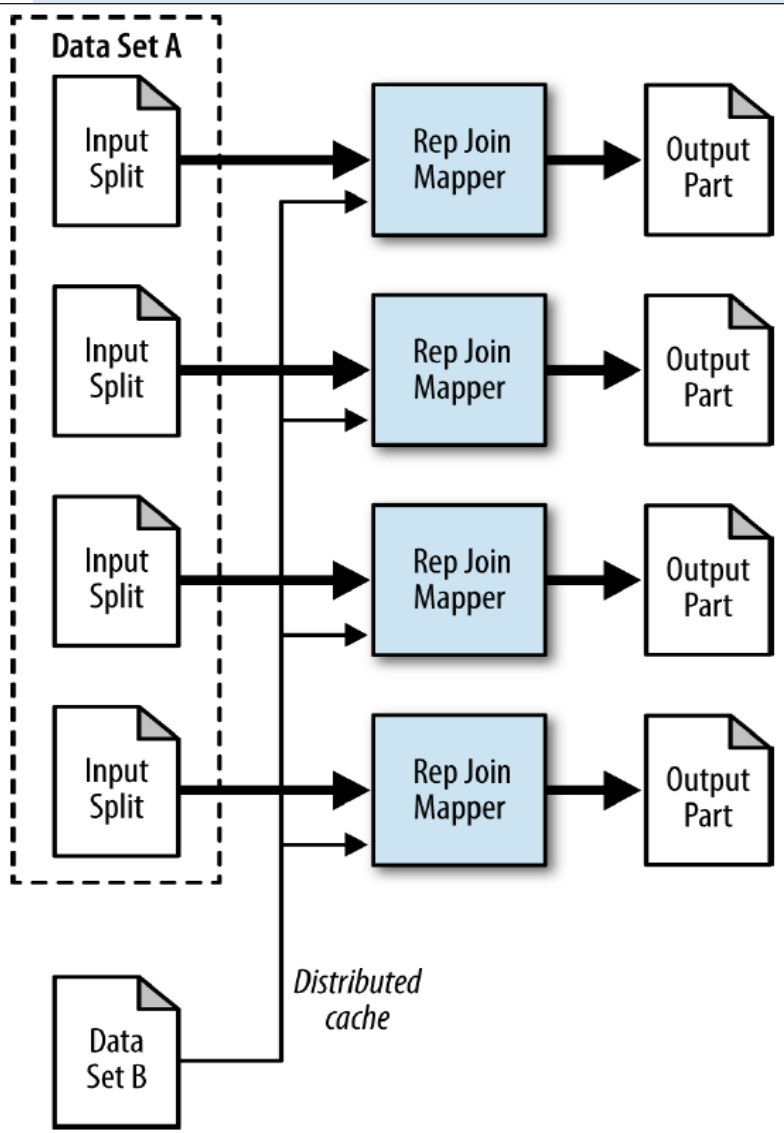
  ON T1.ID=T2.ID

- Performance analysis
  - # of part files == # of reduce tasks
  - Heavy network traffic
  - Utilize relatively more reducers than your analytic
- Ex: ReduceSideJoinDriver.java
  - In-1: Users.xml
  - In-2: Comments.xml

Replicated Join 1/3

- Intent
  - Eliminates the need to shuffle any data to the reduce phase
- Motivation
  - All the data sets except the very large one are essentially read into memory during the setup phase of each map task, which is limited by the JVM heap
- Applicability
  - All of the data sets, except for the large one, can be fit into main memory of each map task

Replicated Join 2/3



- Structure
  - Map-only pattern
  - Read all files from the distributed cache and store them into in-memory lookup tables

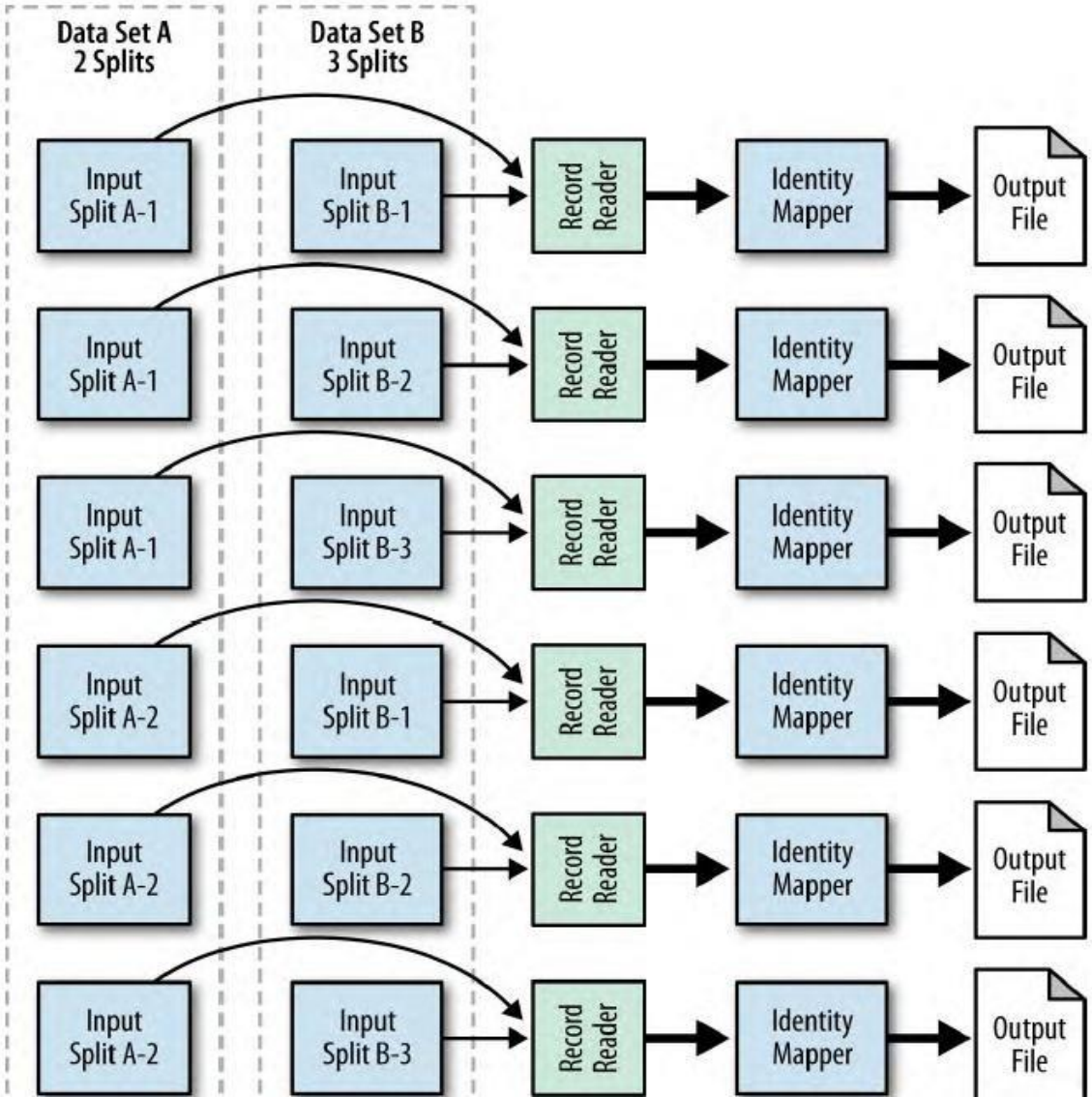
Replicated Join 3/3

- Consequences
  - # of part files == # of map tasks
  - The part files contain the full set of joined records
- Performance analysis
  - A replicated join can be the fastest type of join because there is no reducer required
  - The amount of data that can be stored safely inside JVM
- Ex: ReplicatedJoinDriver.java

Cartesian Product 1/3

- Intent
  - Pair up and compare every single record with every other record in a data set
- Motivation
  - Simply pairs every record of a data set with every record of all the other data sets
  - To analyze relationships between one or more data sets
- Applicability
  - You want to analyze relationships between all pairs of individual records
  - You've exhausted all other means to solve this problem.
  - You have no time constraints on execution time

Cartesian Product 2/3



- Structure
  - Map-only
  - RecordReader job

Cartesian Product 3/3

- Consequences
  - The final data set is made up of tuples equivalent to the number of input data sets
  - Every possible tuple combination from the input records is represented in the final output
- SQL

  SELECT * FROM tableA, tableB;
- Performance Analysis
  - A massive explosion in data size O(n^2)
  - If a single input split contains a thousand records, the right input split needs to be read a thousand times before the task can finish
  - If a single task fails for an odd reason, the whole thing needs to be restarted
- Ex: CartesianProduct.java

References

- Donald Miner and Adam Shook, *MapReduce Design Patterns*.
  - http://oreil.ly/mapreduce-design-patterns
  - https://github.com/adamjshook/mapreducepatterns