CS 378 – Big Data Programming

Lecture 13

Join Patterns

Review

- Assignment 5 – User Sessions

- We'll look at implementation details of:
  - Parsing logs
  - Avro schema
  - Populating Avro object with data
  - Mapper
  - Combiner
    - Should we use one? Can we use one?
  - Reducer

Assignment 6

- Reduce-side join of impression stats for VINs

- `MultipleInputs` (multiple mappers)
  - One reads sessions and collects stats
  - Another reads stats data from another source

- An Avro "union" schema is provided

Assignment 6

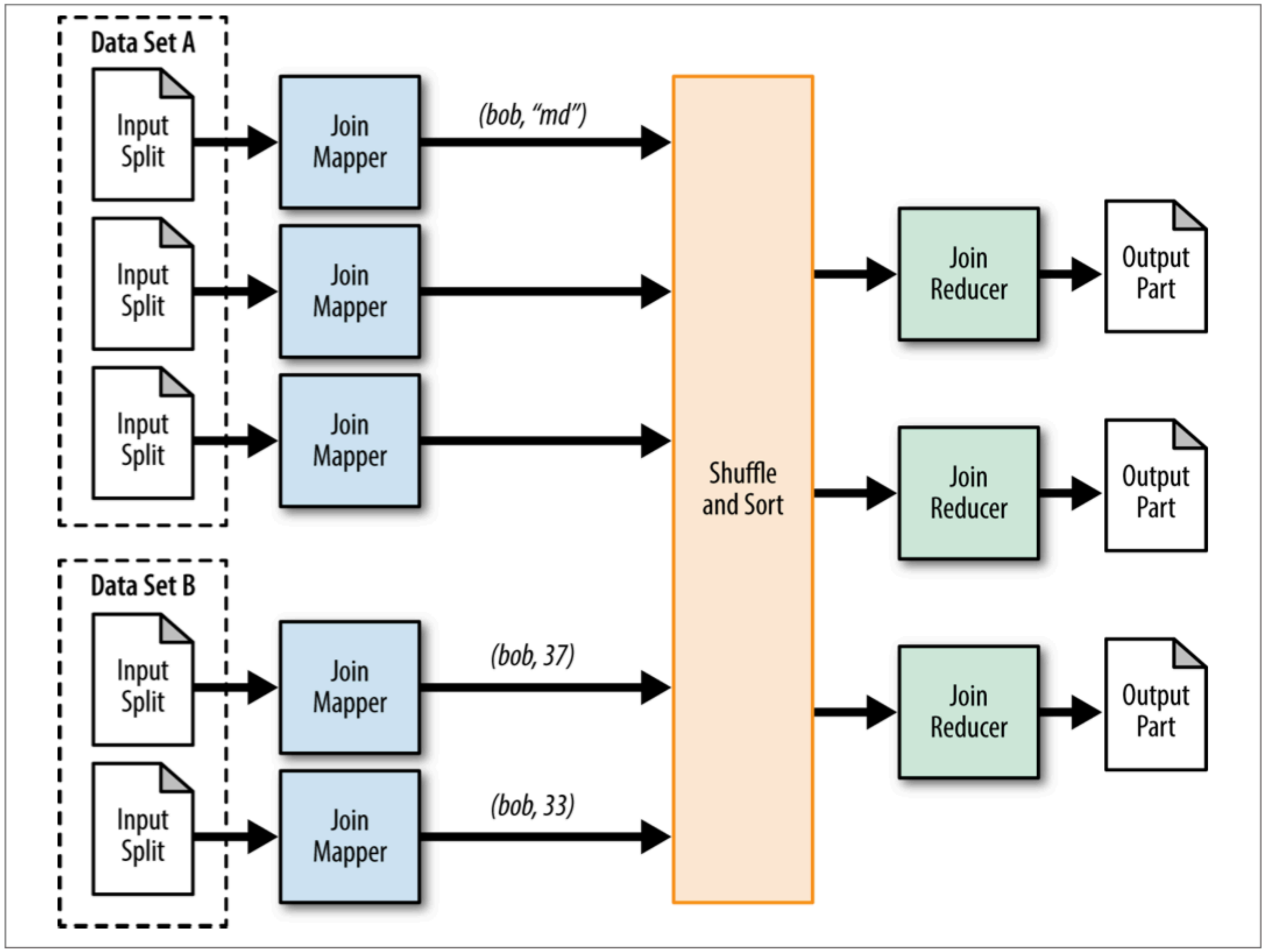
- Modify SessionGenerator
  - Output sessions to AVRO container file (binary)
  - Use **`AvroKeyValueOutputFormat`**
- Define a mapper to read sessions
  - Use **`AvroKeyValueInputFormat`**
- Define a mapper to read data file (CSV)
- Associate mappers with input files
  - Use **`MultipleInputs`**

Join Patterns

- Suppose we only wanted sessions with submits
  - In practice, a small % of sessions have submits
- In our current implementation, we can't identify these sessions until we "reduce" them
- How could we avoid transferring all the impressions for no-submit sessions from mappers to reducers?
  - Mappers would need to know which log entries to ignore

Reduce Side Join - Data Flow

Figure 5-1 from MapReduce Design Patterns



Join Patterns

- Could we tell each mapper which userIds to accept?

- First we'll need to get that info to each mapper
  - Somehow we'll need to get some info to all mappers
  - A list of userIds?

- We still have an issue if that list is too large to hold in memory

DistributedCache

- The Hadoop class: `DistributedCache`
- Allows us to specify files that are distributed to the local file system of each task (mapper or reducer)
- What do we do about the file/data size?
  - Could still be too large to hold in memory

DistributedCache

- In the driver code (`run()` method)
  - Get the file name from the command line
  - Tell Hadoop about this file
  - Name(s) conveyed in the configuration object

```
Path userIdsPath = new Path(args[1]);
FileStatus[] files =
   FileSystem.getConf().listStatus(userIdsPath);
DistributedCache.addCacheFile(
   files[0].getPath().toUri(), conf);
```

DistributedCache

- In the mapper code (`setup()` method)
  - `setup()` method called once for each mapper
  - Get the file name from the configuration
  - Load info from the file(s)

```
URI[] files = DistributedCache.getCacheFiles(
    context.getConfiguration());
```

- What do we do about file/data size?

Bloom Filter

- Probabilistic data structure
  - Used to test whether something is in a predefined set
  - Can create “false positives”
    - Knows for sure that something is not a member of the set
    - Sometimes reports membership as true, when it is false
  - Never creates “false negatives”
    - Never reports “not a member” when it in fact it is a member

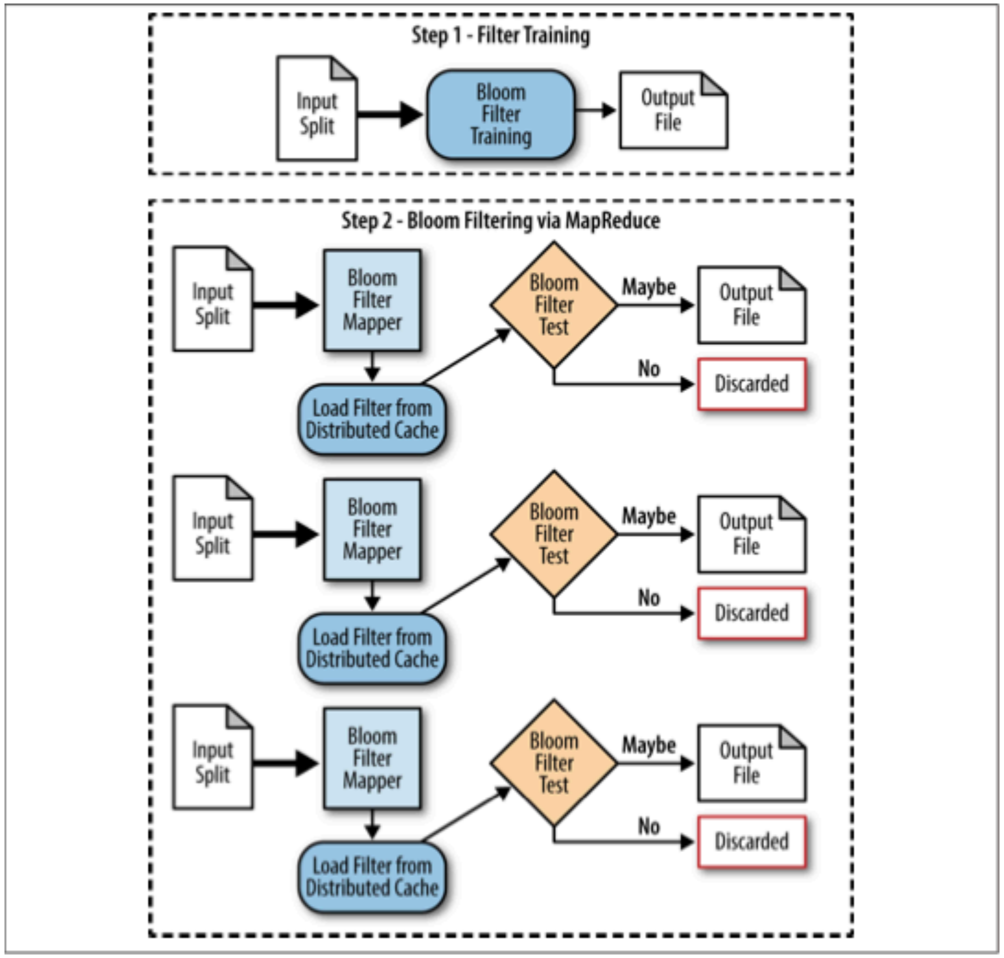
- Fixed size in memory
  - Train the filter using members of the set

Bloom Filter

- Can add members to the set (further training)
  - Can't remove members
  - There is a technique that allows removal
- Parameters of the filter
  - Number of bits in a bit array
  - Number of independent hash functions
- These can be tuned to get a certain false positive rate

Bloom Filter – Data Flow

Figure 3-2 from MapReduce Design Patterns



Reduce Side Join with Bloom Filter

- Train the filter
  - Read all log entries, identify userIds with submits

- Specify the trained data file in our driver app (run() method)

- Modify the mapper to load the trained Bloom filter
  - setup() method

- Reducer – what does it need to do?