CS 378 – Big Data Programming

Lecture 25

Caching, Partitions

Review

- Assignment 11
  - Create user sessions
  - Order events by timestamp, event type, subtype
  - Order sessions by user ID
  - Partition sessions by referring domain
  - Sample SHOWER sessions (1 in 10)

Partitioning Review

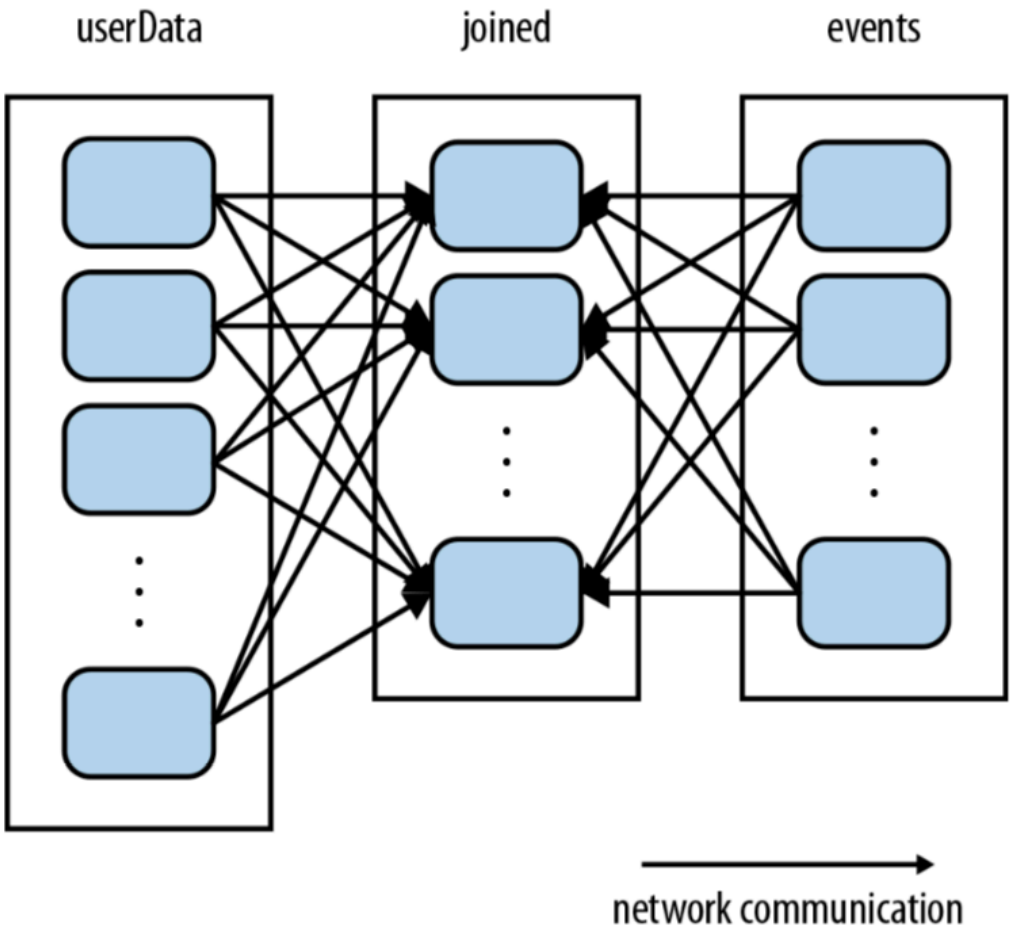
- Partitioning on pair RDDs (key, value)

- Consider an RDD containing user sessions
  - All users over some time period (day or week)
  - We want to merge in the last hour of events

- We'll merge sessions and events by userID

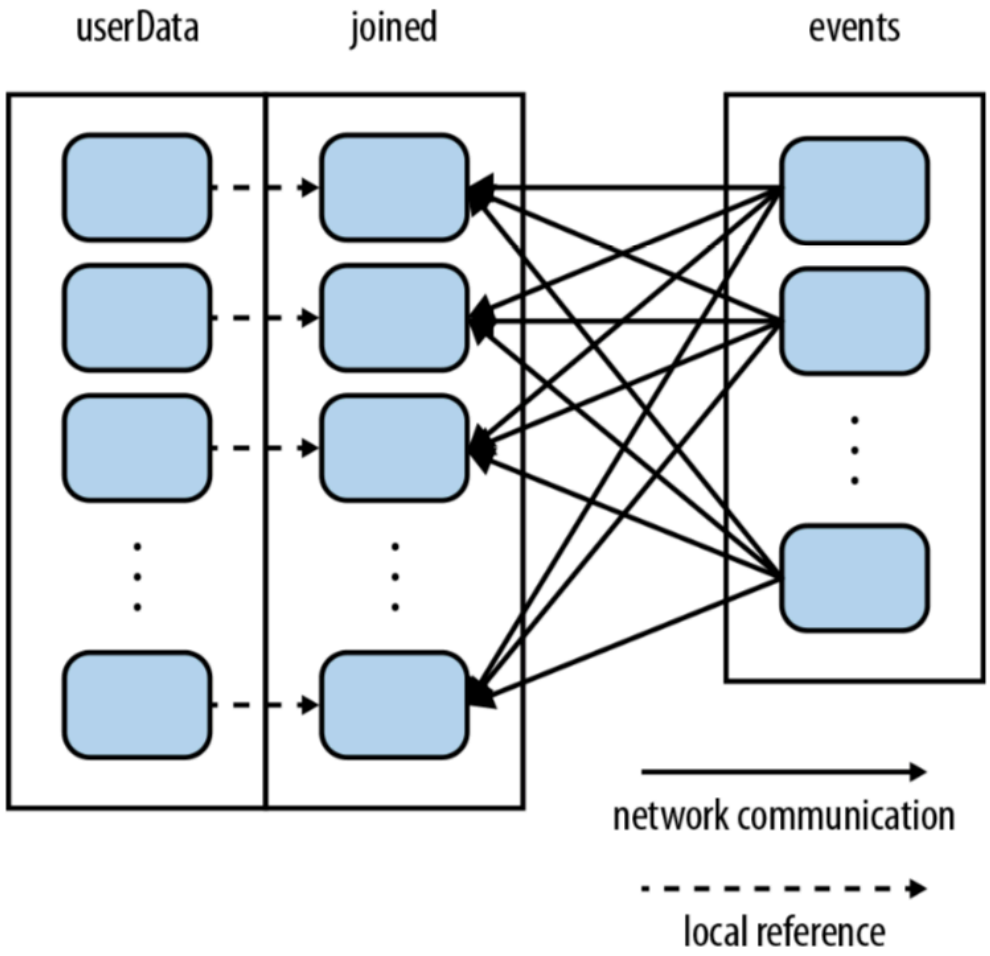
Partitioning Review

Figure 4-4, from Learning Spark



Partitioning Review

Figure 4-5, from Learning Spark



Partitioning

- Consider an RDD containing user sessions
  - All users over some time period (day or week)
  - We want to merge events, multiple times

- To set up for this:
  - Create the session RDD (reading from HDFS)
  - Partition (call `partitionBy()`, a transformation)
  - Persist

Partitioning

- Once an RDD is created with `partitionBy()` or another transformation that implicitly partitions,

- You should persist the RDD, otherwise the partitioning will be repeated on the next action

Benefits of Partitioning

- Many transformations shuffle data across the network
- All these will benefit from partitioning
  - `cogroup()`
  - `groupWith()`
  - `join()`
  - `leftOuterJoin()`
  - `rightOuterJoin()`

Benefits of Partitioning

- And these will benefit from partitioning
  - `groupByKey()`
  - `reduceByKey()`
  - `combineByKey()`
  - `lookup()`

Benefits of Partitioning

- Transformations on a single, partitioned RDD
  - Computed locally on a machine
  - Reduced result is sent to the master machine

- Binary transformations like `cogroup(), join()`
  - Prepartitioning will cause one RDD not to be shuffled
  - If both RDDs have the same partitioner and are on the same machine (e.g., from `mapValues()`)
  - No shuffling will occur

Partitioning

- Some transformations automatically return an RDD with known partitioning
- `sortByKey()` – range partitioned
- `groupByKey()` – hash partitioned

- Some transformations "forget" parent partitioning
  - `map()`

Partitioning

- Which partitioner is set on output?

- Depends on the parent RDDs' partitioners
- By default, hash partitioner
  - Number of partitions is the level of parallelism
- If one parent has an explicit partitioner
  - Use it
- If both have an explicit partitioner, use the first

Partitioning

- To maximize the potential for partitioning-related optimizations, instead of map() use
- `mapValues()`
- `flatMapValues()`

- Why? They preserve the key

Custom Partitioners

- Partitioners used by default:
  - `HashPartitioner`
  - `RangePartitioner`

- Custom partitioner
  - Subclass `Partitioner`
  - Implement the required methods
    - `numPartitions()`
    - `getPartition(key)`
    - `equals()`