CS 378 – Big Data Programming

Lecture 24

More on Partitions

Accumulators

Review

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

Review

- Caching/persistence
  - Can delete/remove using: `unpersist()`

- Java instances as closures
  - Can reference static variables defined in the enclosing lexical scope
  - Can reference final local variables of the enclosing method

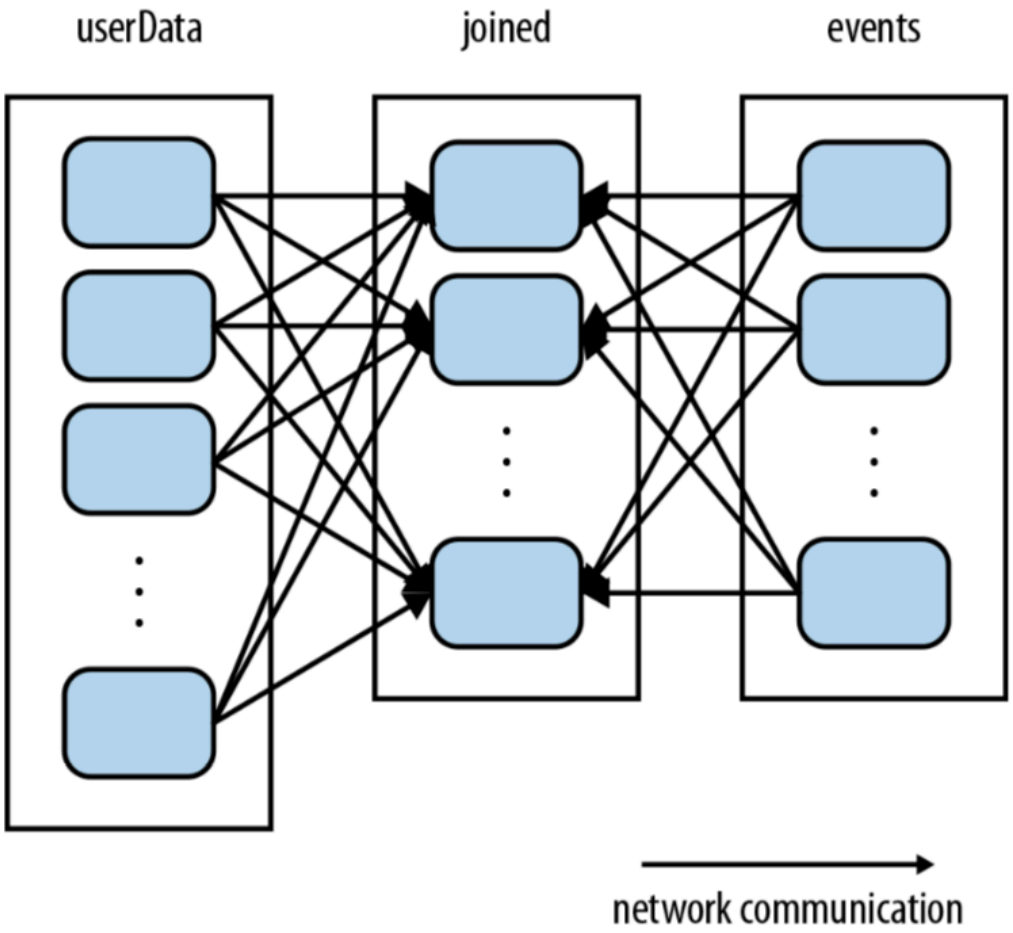
Partitioning

- Prudent partitioning can greatly reduce the amount of communication (shuffle)

- If an RDD is scanned only once, no need
- If an RDD is reused multiple times in key-oriented operations
  - Partitioning can improve performance significantly

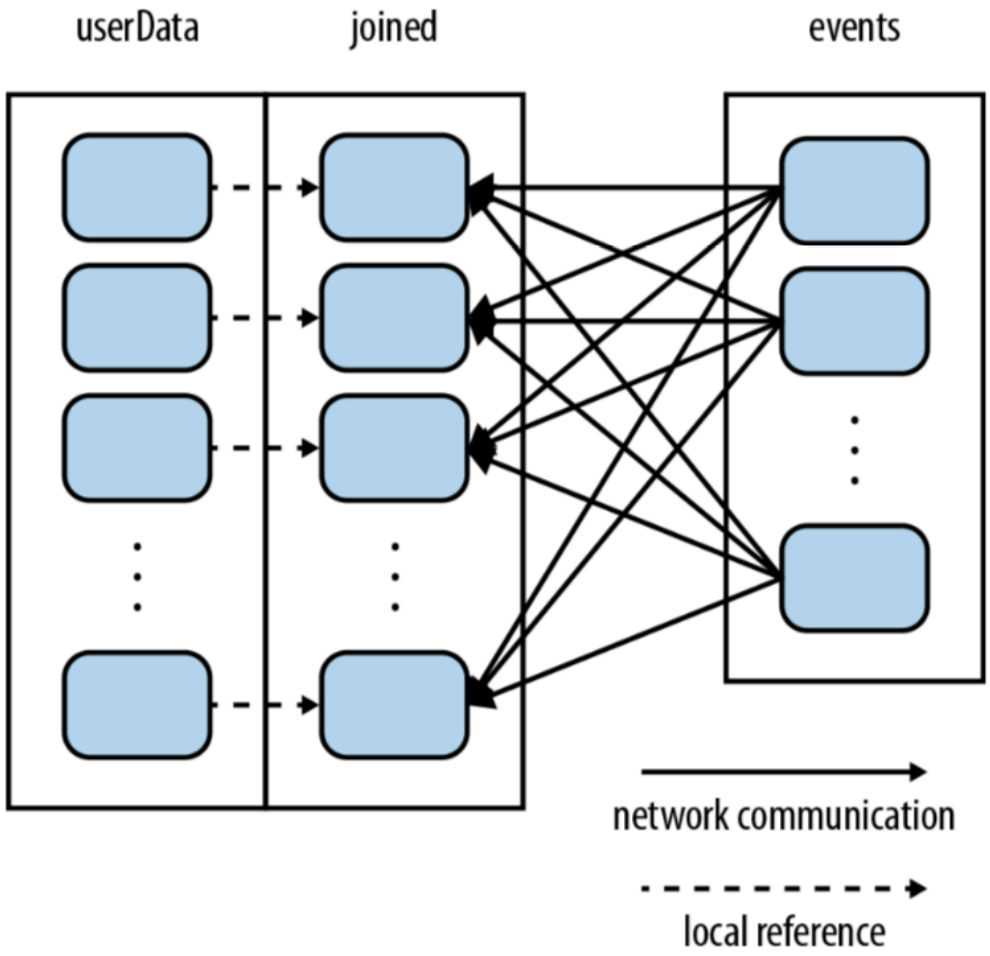
Partitioning

Figure 4-4, from Learning Spark



Partitioning

Figure 4-5, from Learning Spark



Partitioning

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

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

Partitioning

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

- Some transformations "forget" parent partitioning. Why?
  - `map()`

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
  - A 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

- 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()`

Accumulators

- In our session generator app,
- Suppose we wanted to count the number of sessions that are sampled (SHOWER, 1 in 10)

- How would we do this?

- How did we do this using Hadoop map-reduce?

Accumulators

- An accumulator provides a means for aggregating values from worker nodes back to the driver node.
- Create an accumulator from the Spark context
- Increment the accumulator in functions passed to worker nodes

Accumulators

- For failures or re-evaluation, what happens?

- Actions:
  - Each task's update applied only once

- Transformations:
  - No guarantee that task updates applied only once
  - Re-evaluation will update accumulator each time