Sorting in Spark is a involves multiple steps, generally:
Sampling -> input RDD is sampled and is used to compute boundaries for each output partition
Partitioning -> input RDD is partitioned using a partitioner (in my case, I chose to leverage timestamp_detected and a HashPartitioner. There are other ways to partition, such as a RangePartitioner)
Shuffling -> data is moved between nodes to achieve the desired order

We can think of the sorting strategy in Spark (and distributed computation in general) as a divide-and-conquer approach.
(1) We sample and "fan-out" the data (smaller chunks of the data called partitions are spread across nodes).
(2) We apply an algorithm such as MapReduce in parallel across the nodes (based on the number of executors, which is bound by the compute resources, eg. CPU cores)
(3) Each executor sends its results back ("fan-in") to the driver (communication is coordinated by the cluster manager). If a global order is required (eg. Join then order by), the data is redistributed such that rows with similar keys are located in the same partition. This also implies that if data skew is not handled, Spark may not be very efficient since the nodes and executors may not be utilized.

My approach was to rely on the fact that the data from Parquet file 2 is much smaller. So we broadcast this data (which is a lookup map for geolocation names) across the nodes. Each node is now able to quickly augment its own rows using this lookup map that is in-memory.
Refer to App.getVideoCameraItemsDetectedByLocationRdd() method.