```
/* Scala/Java RDD Distance join query */
JoinQuery.DistanceJoinQuery(PickupPointRDD, StopStationRDD, 1)
```

***GSJoin* ALGORITHM** GEOSPARK combines the approaches proposed by [42–44] to a new spatial range join algorithm, namely *GSJoin*, that re-uses spatial partitioned RDDs as well as their indexes. This algorithm, which takes two spatial partitioned RDDs A and B (i.e., TaxiStopStations and TaxiTripTable), consists of three steps as follows (see Algorithm 4 and Fig. 4).

- **Zip partitions** This step zips the partitions from A and B (TaxiStopStations and TaxiTripTable) according to their grid IDs. For instance, we merge Partition 1 from A and B to a bigger partition which has two sub-partitions. Both Partition 1 s from A and B have the spatial objects that fall inside Grid 1 (see Section 4.3). Partition 1 from A (TaxiStopStations) contains all taxi stop stations that locate in Grid 1 and Partition 1 from B (TaxiTripTable) contains all taxi trips that are picked up in Grid 1. Note that, the data in Partition1 from A is guaranteed to disjoint from other partitions (except 1) of B because they belong to totally different spatial regions (see Section 4.3). Thus, *GSJoin* does not waste time on checking other partitions from B with Partition 1 from A. This Zip operation applies to all partitions from A and B and produces an intermediate RDD called C.
- **Partition-level local join (no index)** This step runs a partition-level local range join on each partition of C. Each partition from C has two sub-partitions, one from A and one from B. If no indexes exist on both the sub-partitions, the local join will perform a nested loop join that traverses all possible pairs of spatial objects from the two sub-partitions and returns the qualified pairs. This costs $O(n^2)$ complexity on each C partition, where n is the number of objects in a sub-partition.
- **Partition-level local join (with index)** During the partition-level local join step, if an index exists on either one sub-partition (say, sub-partition from B is indexed), this local join will do an index-nested loop. It uses each object in the sub-partition from A as the query window to query the index of the sub-partition from B. This costs O(n*log(n)) complexity on each C partition, where n is the number of objects in a sub-partition. It is worth noting that this step also follows the Filter and Refine model which is similar to this part in Range query and Distance query (mentioned in Section 4.4). Each query

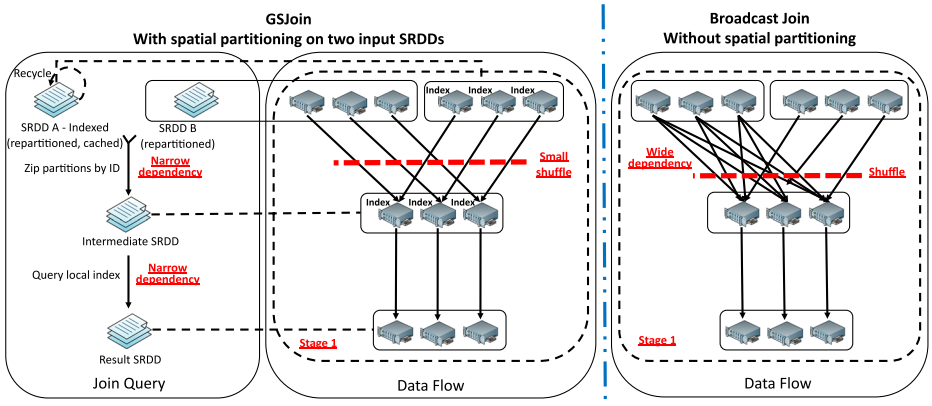

Fig. 4 Join query DAG and data flow