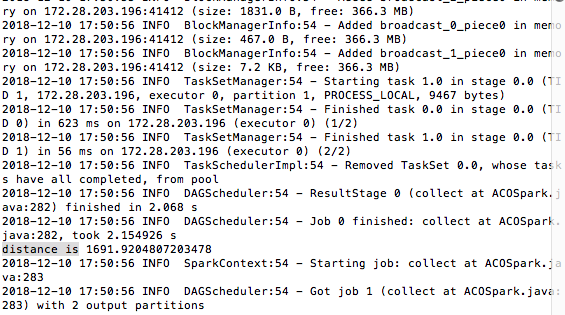
Parallelization Technique:  
In spark version of the project. We took a different approach. In the sequential version of the ant colony optimization. We instantiated a couple 2 dimensional arrays. The natural thing to do is to create a RDD for each matrix. Each of them will be updated through out the program, values in some of the matrices are dependent on other matrices. The approach I took was that I created a new Ant class that implements the Serializable class. Each Ant carries a route array that stores the route for individual ant, a ant ID number to store in ROUTES 2d array, as well as the distance of the route stored in route array. I then created a list of Ant, the size is the number of ants we want to run. I used Spark's parallelize() method to convert the list into a RDD. Now that we have a initial JavaRDD. We can start the optimization. I put all my optimization into a while loop, the loop terminals when the number of interactions we preset has been reached. In the while loop, I first broadcast my Pheromone matrix using Spark broadcast function. This is important because each iteration, the Pheromone is updated using the distance of each ant calculated from previous iteration. But in Spark, we cannot simply store the Pheromone in a global matrix and update them in the RDD. I first transform my ant RDD to a PairRDD where they key is the distance calculated, and the value is a Ant class with updated value. Inside the transformation, I let each ant calculated their own route and calculate the distance. I record the route and distance and return the Tuple. Now that I have a tuple RDD where the key is the distance, I can use RDD sortByKey function to sort the RDD. I obtain the best result from all ants by simply collect the RDD and get the first element. I use a global bestRoute variable to keep track of the best result so far. I then update the pheromone. Note that in updatePheromone function, we need the distance from all Ants to calculate delta Pheromone. We store that into a temporary array called currentLengths. The last transformation is to reset all Tuples in the RDD to original states: set each element in route array for each ant to -1, set their distance to 0, etc.

Execution Performace:

Unfortunately, with 1000 iterations, I only get to 1692 total distances as best route, I let the Spark program run longer and did not obtain any better results. I suspect that this is due to the fact the broadcast function may be slowing the program down by a lot. Every iteration, it needs to be recalculated to rebroadcast to all nodes. In my implementation: I let each RDD element calculate its own route, and I collect the RDD element in every iteration. there are a lot of transforming and collecting over the course of the program, it will slow down the calculation even further.

Programability:

Our specific implementation of the ACO may not be the best fit to be run on Spark. In general, one main advantage of Spark is the amount of boiler code can be fairly small due to the advantage of using Lamda expression. But in this program, I actually had to write more lines of code than the sequential version. The extra boilerplate code comes mainly from the work to rebroadcast Pheromone. I wrote code to broadcast it, collect distances from all ants, and then rebroadcast it after updating the Pheromone. This is because in Spark, we cannot update global data structures inside of RDD. We can read them after broadcasting it. To change a shared data structure across different nodes. I had to collect data needed to update the Pheromone in every iteration before updating it, and then rebroadcast.

Application Summary:

For project 5, we implemented the Ant Colony Optimization of Traveling Salesman problem. The Ant Colony Optimization imitates real ant colonies. When ants look for food on the ground. a group of ants will each explore different route. If they find food, on their way back, they release Pheromones to mark the route for later searchers. When more ants explore a good routes, Pheromones on the route becomes more intense. Routes with heavier Pheromones will take precedence over routes with lighter Pheromones in the future. To encourage ants to explore different route and avoid local optima, we introduce a evaporation rate to adjust Pheromones at each iteration. We use these information to calculate a probability for each ant to move from city i to city j using the following formula:

There are three ways to update Pheromones in Ant Colony Optimization: Ant Cycle, Ant Density and Ant Quantity. We used the Ant Cycle model as it is more straight forward. In Ant Cycle model, the change in Pheromone is calculated by diving the current Pheromone by the total distance of the route calculated for current iteration if the ant passed through a given city pair, if the ant did not pass through the city pair, there is no change in Pheromone for this iteration.