CS744A3 Part B

Group 21

September 2017

Application 1. Question 1. Implementation of the PageRank algorithm, after loading the graph, I first construction another graph with the vertices data as their outdegrees

```
val outDegGraph = graph.outerJoinVertices(graph.outDegrees){
(_, _, degOpt) => degOpt.getOrElse(0)}.persist()
```

And initialize the ranks as a VertexRDD.

```
var ranks = graph.vertices.mapValues(r => 1.0)
```

In each iteration, I construct an outputGraph with the source attribute to be the contribution,

```
val outputGraph = outDegGraph.outerJoinVertices(ranks){
(_, deg, rank) => rank.getOrElse(0.15) / deg}
```

and use aggregateMessages to sum up the contributions

```
val contri = outputGraph.aggregateMessages[(Double)](
triplet => { triplet.sendToDst(triplet.srcAttr) },  // map
_ + _   // reduce)
```

The complete time, network/disk bandwidth and number of tasks are summarized in Table 1, 2.

The benefit of GraphX, as the original papers says, is the Flexible vertex-cut partitioning. This is verified by the experiment result. The GraphX has an obviously smaller network transmit than the general RDD, meaning that it has a much better graph partition.

Application 2. Build the graph, we take the input of A2P2Q2, build vertices from intervals and remove high frequent words (tha show up in more than 50% of all the intervals) so that we get a graph with 359 vertices and 14928 edges. We output the graph to two files. The first one is the edges with each line a (src, dst) pair. The second is the vertices with each line the set of words in the interval. When reconstructing the graph in Spark, we first read edges using GraphLoader.edgeListFile, and read vertices as an RDD[(VertexId, Array[String])]. Then we outerjoin the vertices into the graph.

Question 1. Just filter the triplets and count the valid triplets.

```
val res = graph.triplets.filter {
triplet => triplet.srcAttr.length > triplet.dstAttr.length
}.count()
```

Question 2. Use outerjoin to create a new graph with VD a (outdegree, numberOfWords) pair. Then reduce the vertices

```
val res = outdegAndSizeGraph.vertices.reduce {
(a, b) => if (a._2._1 > b._2._1 || (a._2._1 == b._2._1 && a._2._2 > b._2._2)) a else b
}
```

hosts	Completion time (min)	Network Transmit Size (MiB)		Disk Bandwidth (MB/s)		Number of tasks
		Receive	Transmit	Read	Write	
vm-21-1	2.6	177.92	5.56	4.84	0.45	352
vm-21-2		791.61	716.53	8.21	2.47	
vm-21-3		2661.40	2491.16	3.49	2.26	
vm-21-4		2588.60	2688.18	13.23	2.14	
vm-21-5		736.91	791.63	10.26	3.39	

Table 1: Qeustion 1 Page Rank using GraphX

hosts	Completion time (min)	Network Transmit Size (MiB)		Disk Bandwidth (MB/s)		Number of tasks
		Receive	Transmit	Read	Write	
vm-21-1	8.5	8.59	5.19	3.13	0.32	427
vm-21-2		5208.53	7583.68	4.85	2.62	
vm-21-3		3965.05	2367.81	5.89	1.95	
vm-21-4		4908.49	7242.59	5.88	2.33	
vm-21-5		5960.51	2145.32	1.59	2.49	

Table 2: Qeustion 1 Page Rank using General RDDs

Question 3. Aggregate messages

```
val neighborSizes = graph.aggregateMessages[(Int, Double)] (
triplet => {triplet.sendToDst(1, triplet.srcAttr.length)},
(a, b) => (a._1 + b._1, a._2 + b._2)
)
```

and compute the average

```
val averageSize = neighborSizes.mapValues(
(id, value) => value match { case (count, totalNum) => totalNum / count}
)
```