Big Data System Midterm Exam

November 10, 2013

1 (20%) Term Explanation

1. (5%) What is combiner? How can it help with the performance of MapReduce under the scenario of big-data?
2. (5%) In Spark, what's the difference between Transformation and Action?
3. (5%) What is aggregator in Pregel? How can it be utilized? Please give at least 2 examples.
4. (5%) What does "defined" means in GFS? (Hint: compare with the definition of "consistent" in GFS)

2 (20%) Map-Reduce

In statistics, logistic regression a type of probabilistic classification model . Given large scale of data point, the format of each data point is as below. Given the original Logistic Regression and MapReduce Example(WordCount). Please rewrite the logistic regression algorithm into MapReduce.

$$\text{m data point} \left\{ \begin{array}{l} X_{N1}Y1 \\ X_{N2}Y2 \\ \ldots \\ X_{Ni}Yi \\ \ldots \\ X_{Nm}Ym \end{array} \right\} \begin{array}{l} \text{(X\_N is N-Dimensional Vector)} \\ \text{(Y is Label)} \end{array} \quad (1)$$

```
w = Vector.Random(N) // N-dimensional vector
for i from 1 to ITERATIONS do{
  //Compute gradient
  g =Vector.Zero(N) // N-dimensional zero
    vector
  for every data point (XN, Y) do {
    //XN is a vector, Y is +1 or 0
    g += (Y * XN) / (1 + exp(Y * MatrixMulti(w
     * XN)))
  }
  w -= LEARNING_RATE * g
}
```

Listing 1: Logistic Regression Algorithm

```
main()
{
  runJob()
}

map(key,value)
{
  //key:document name
  //value:document content
  for each word w in value:
  EmitIntermediate(w, "1")
}

reduce(key,value)
{
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
  result += ParseInt(v);
  Emit(AsString(result));
}
```

Listing 2: WordCount in MapReduce

3 (20%) Google File System

1. (10%) Please give **2** brief descriptions of the trade-off of chunk size in GFS.
2. (10%) Please give a brief description of how GFS achieve fault tolerance (**both** high availability and data integrity).

4 (20%) Spark

Most MapReduce systems are built around an acyclic data flow model that is not suitable for iterative applications.Therefore, Spark propose an in-memory computing framework that supports these applications while retaining the scalability and fault tolerance of MapReduce.

1. (5%) Please explain why MapReduce framework is infeasible when it comes to iterative computations.
2. (5%) Does Spark always outperform Hadoop MapReduce framework? If no, please give an example and briefly explain it.
3. (10%) What's the advantage of Lineage when it comes to fault tolerance on big data computation. (Hint: compare with traditional approaches like logging / checkpoint)

5 (20%) Bulk Synchrounous Parallel

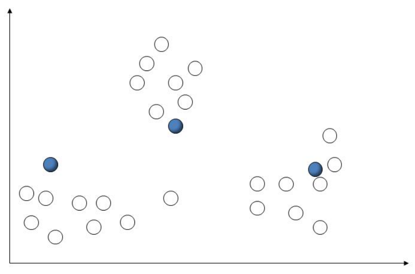
1. (6%) Please describe what the Bulk Synchronous Parallel (BSP) model is in detail, i.e. you must give detail explanation of what each phase is doing.
2. (8%) Describe how graph algorithms can be mapped to the BSP model, including how the computation terminates.
3. (6%) Single source shortest path is an important problem in graph computing. Given a graph and a starting vertex, we want to find the shortest path to every node other than the starting node. Please devise an algorithm using the BSP model that calculates the shortest "distance" to every node other than the starting node. The graph has only positive edges and edges might have different values. You have to describe your algorithm in **pseudo code**. Note that we only care about the shortest distance so you don't have to find the actual path.

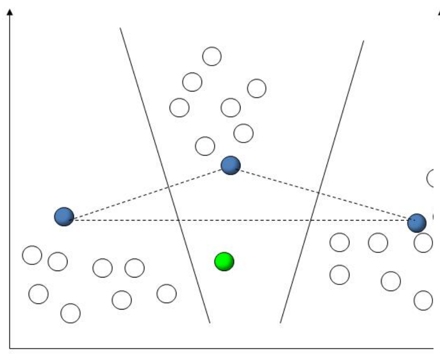
6 (20%)(Bonus) K-Means in Spark

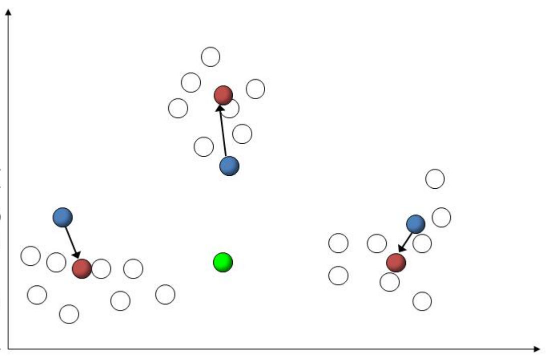
In machine learning, the K-means algorithm is very useful for solving unsupervised clustering problem, which tries to partition a set of data points into K groups where K is a given number. The algorithm is simplified as following:

1. Given K, the number of groups we'd like to partition the D-dimensional data point set X into.
2. Randomly generate K points, $C_{0\sim(K-1)}$, as centroids of the groups.
3. Each point in X take the id of it's nearest centroid as it's group label
4. Update C_i as the arithmetic mean of all points taking C_i as it's centroid
5. Repeat 3 & 4 until iteration limit exceeds or convergence(centroids not changing much)

However, the computation on finding centroid would be a great bottleneck when we have enormous amount of data points, and the iterative intrinsic is make it infeasible for using MapReduce framework. So in this section, you're asked to


(a) initialize centroids


(b) cluster by centroid


(c) update centroids

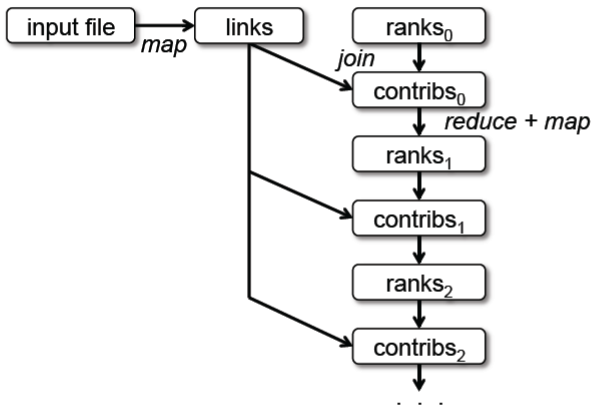


Figure 2: PageRank Lineage

1. (5%) Draw out the Lineage, like something in Figure 2.
2. (15%) Write Spark Driver Program ***psuedo code*** for K-means here and briefly explain your design.

Again, PSEUDO-CODE is enough, but your answer must be clearly commented and preserve critical properties of Spark. (Operation granularity, RDD creation, Broadcast variable)

Here are some assumptions:

1. For simplicity, just print out the clustering result as [LABEL POINT_CONTENT]. e.g.
   1 0.882 0.24 0.25213...
   1 0.1238 1.8102 1230.33...
   2 3.212 223.1 120.12
   ...
2. Data is provided as a file: each line, consisting of D fields of real numbers corresponding to the value of the a coordinate respectively in order, represents a data point.
3. K, D, and ITERATION_LIMIT are predefined, meaning that you can use them directly.
4. Use Euclidean distance as your distance measure.
5. `euc_dist(a, b), avg(point_lists), closest(centroids_list, p)` are built in; use them directly.
6. Mark each cluster with integer IDs.

7. Although in Spark you can use ”`for`” with slightly different syntax to apply operations on an object parallelly, please use `for_each( data ) ...` instead for such case. It’s helpful if you don’t know names of certain built-in operations, but use it carefully. In fact, you can finish this section without it.

8. Make your own trivial assumptions (ex: file name)

Hints

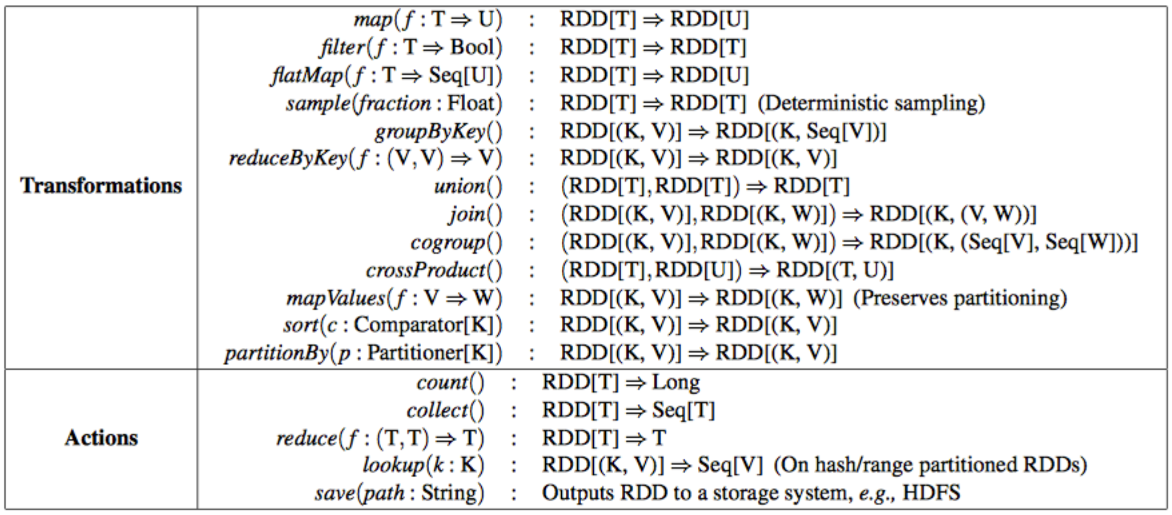
1. You can invoke operations on RDDs in chain. For example:
```
var a = read_from_file("sample.txt") // [1,2,3,4,5]
var b = a.map(i => i*2).map(i => i*4) // [8,16,24,32,40]
```

2. K is very small, say 5 or 10.

3. In Spark, if you have some information that should be shared among workers, say, a look up table for example, you should make it a *Broadcast variable.* e.g.
```
var lookup_table = parse_graph_from_file("graph.txt")
lookup_table = spark.braodcast(lookup_table)
```

4. `groupByKey()` is helpful.

5. Figure 3 is a table of transformations and actions of RDD. Again, pseudo code is fine.



Transformations	*map*(f : T $\Rightarrow$ U)	: RDD[T] $\Rightarrow$ RDD[U]
	filter(f : T $\Rightarrow$ Bool)	: RDD[T] $\Rightarrow$ RDD[T]
	flatMap(f : T $\Rightarrow$ Seq[U])	: RDD[T] $\Rightarrow$ RDD[U]
	sample(*fraction* : Float)	: RDD[T] $\Rightarrow$ RDD[T] (Deterministic sampling)
	groupByKey()	: RDD[(K, V)] $\Rightarrow$ RDD[(K, Seq[V])]
	reduceByKey(f : (V,V) $\Rightarrow$ V)	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	union()	: (RDD[T],RDD[T]) $\Rightarrow$ RDD[T]
	join()	: (RDD[(K, V)],RDD[(K, W)]) $\Rightarrow$ RDD[(K, (V, W))]
	cogroup()	: (RDD[(K, V)],RDD[(K, W)]) $\Rightarrow$ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	: (RDD[T],RDD[U]) $\Rightarrow$ RDD[(T, U)]
	mapValues(f : V $\Rightarrow$ W)	: RDD[(K, V)] $\Rightarrow$ RDD[(K, W)] (Preserves partitioning)
	sort(c : Comparator[K])	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
	partitionBy(p : Partitioner[K])	: RDD[(K, V)] $\Rightarrow$ RDD[(K, V)]
Actions	*count*()	: RDD[T] $\Rightarrow$ Long
	collect()	: RDD[T] $\Rightarrow$ Seq[T]
	reduce(f : (T,T) $\Rightarrow$ T)	: RDD[T] $\Rightarrow$ T
	lookup(k : K)	: RDD[(K, V)] $\Rightarrow$ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	: Outputs RDD to a storage system, *e.g.*, HDFS

Figure 3: RDD Transformation and Actions