Bisecting K-means

Rafael Jeferson Pezzuto Damaceno

Inteligência na Web e Big Data
Prof. Dr. Fabrício Olivetti de França

Santo André, 11 de Maio de 2018

Base de Dados

Plataforma Acácia

- Banco de dados genealógicos
  - 1.111.544 acadêmicos
  - 1.208.399 orientações
    - Mestrado e Doutorado

Vértice → Acadêmico

Aresta → Orientação formal concluída entre dois acadêmicos

13 métricas

- Descendência + –
- Fecundidade + –
- Fertilidade + –
- Gerações + –
- Orientações + –
- Primos + –
- Índice Genealógico

Base de Dados

Tabela 1. Conjunto de vértices onde o primeiro campo é um identificador do vértice e os restantes são as métricas genealógicas.

id	d+	d-	fc+	fc-	ft+	ft-	p+	p-	gr+	gr-	r+	r-	gi
0	0	6	0	1	0	1	117	0	0	4	0	6	0
1	0	1	0	1	0	0	0	0	0	1	0	1	0
2	0	5	0	1	0	1	71	0	0	3	0	5	0

Parsing e **Normalização** (z-score)

RDD

```
('0',
      [
      -0.08001889587329748,   0.1361842033876333,   -0.2098024552506829 2,
      -0.1681646959022529,    -0.1417237589895481 4,  0.0734087919688399 7,
      -0.2911556035578139 5,  -0.1133942843179100 7,  -0.3777233782245712,
      0.661430471729447 6,    -0.0794409034103 32,    0.1119299239880567 9,
      -0.1714561755206763 8
      ]
), ...
```

Bisecting K-means

BISECTING-K-MEANS(*k*, *numeroTentativas*)
1 Inicialize lista de clusters com todos os pontos
2 **repeat**
3 Remova um cluster da lista de clusters
4 **for** *i* = 1 até *numeroTentativas* **do**
5 Bifurque o cluster selecionado usando **K-MEANS**
6 **end for**
7 Selecione os dois melhores da biseção
8 Adicione os dois clusters à lista de clusters
9 **until** Até que a lista de clusters contenha *k* clusters

Implementação

```
1  def bisectKmeans(data, k, bisects, iteractions):
2      finalClusters = [data]
3      while(len(finalClusters) != k):
4          clusterToSplit = finalClusters.pop()
5          sse = []
6          tmpClusters = []
7          for i in range(bisects):
8              clustersRDD, centroids = kmeans(clusterToSplit, 2, iteractions)
9              tmpClusters.append(clustersRDD.filter(lambda x:x[0] == 0))
10             tmpClusters.append(clustersRDD.filter(lambda x:x[0] == 1))
11             sse1 = getSSE(tmpClusters[-2], centroids[0])
12             sse2 = getSSE(tmpClusters[-1], centroids[1])
13             sse.append(sse1 + sse2)
14         minSseIndex = np.argmin(sse)
15         largerCluster, minorCluster = getClustersByCount(tmpClusters[minSseIndex*2]
16                                                          , tmpClusters[minSseIndex*2 + 1])
17         finalClusters.append(minorCluster.map(lambda x:(x[1], x[2])))
18         finalClusters.append(largerCluster.map(lambda x:(x[1], x[2])))
19     return finalClusters
```

```
1  def getSSE(cluster, centroid):
2      return cluster.map(lambda x:euclidianDistance(x[2], centroid)).reduce(lambda x,y:x+y)
```

Implementação

```
1  def kmeans(data, k, iteractions):
2      centroids = [p[1] for p in chooseRandomPoints(data, k)]
3      for i in range(iteractions):
4          clustersRDD = data.map(lambda x:(np.argmin([euclidianDistance(x[1], c) for c
5                                                      in centroids]), x[0], x[1]))
6          newCentroids = (clustersRDD
7              .map(lambda x:(x[0], [x[2],1]))
8              .reduceByKey(lambda x,y:([(np.array(x[0]) + np.array(y[0])) ,(x[1] + y[1])]))
9              .map(lambda x:list(np.array(x[1][0])/(x[1][1])))
10            ).collect()
11         centroidsHaveChanged = [areCentroidsDifferent(centroids[i], newCentroids[i])
12                                                      for i in range(len(centroids))]
13         if True not in centroidsHaveChanged:
14             return clustersRDD, centroids
15         else:
16             centroids = newCentroids
17     return clustersRDD, centroids
```

Resultados

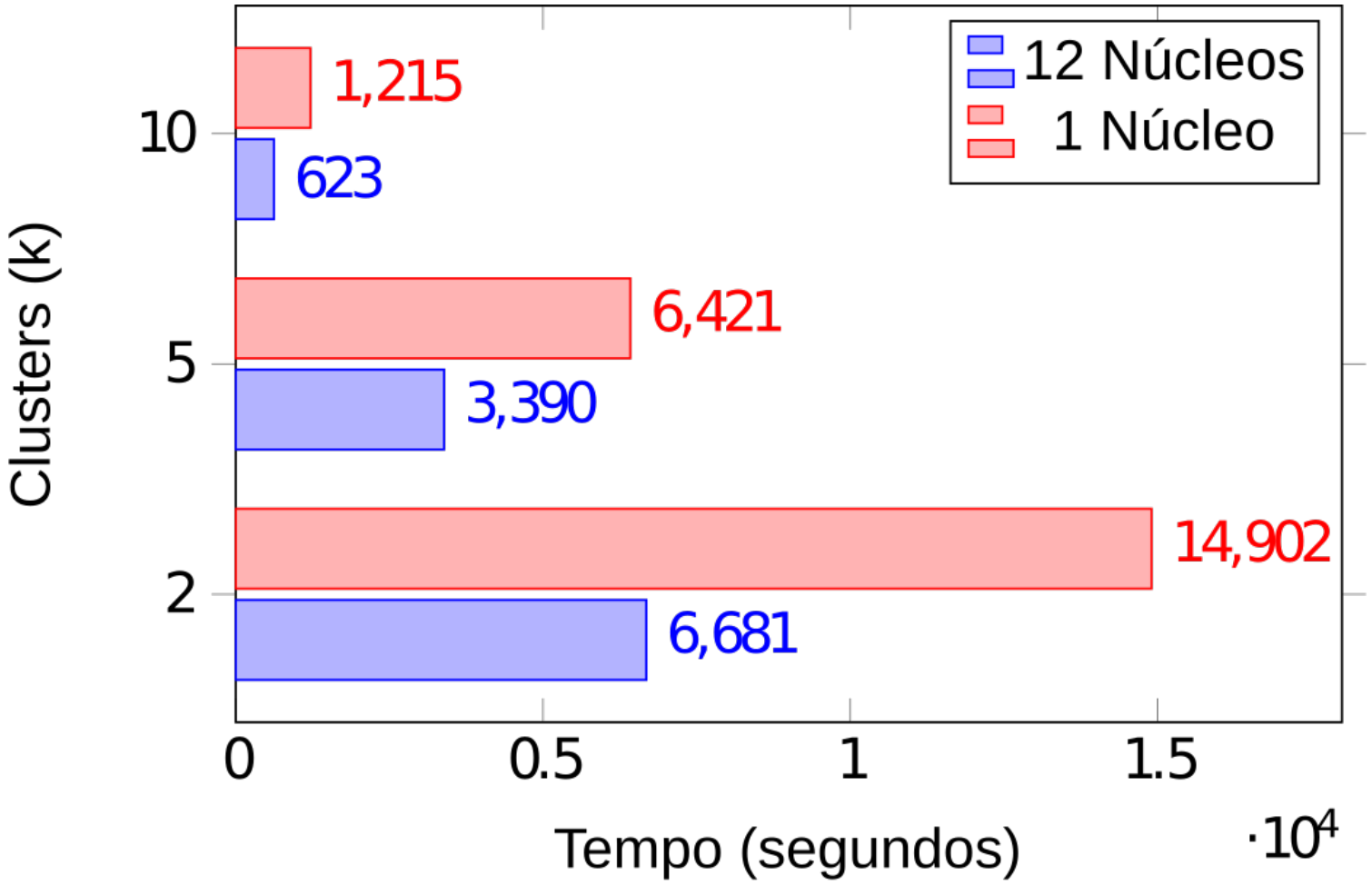

Figura 1. Tempo de execução (eixo x) do algoritmo rodando em paralelo (12 núcleos) e em não paralelo (1 núcleo) para identificar 2, 5 e 10 *clusters* no conjunto de 1,111,544 vértices.