Data Intensive Computing – Handout 4

Training Cluster

You can login to the training cluster *dlrc* via machine `ufallab.ms.mff.cuni.cz` and port 11422, i.e., using `ssh -p 11422 ufallab.ms.mff.cuni.cz`.

All machines in the cluster use same architecture and share `/dlrc_share` directory.

The cluster consists of master `dlrc` and 5 nodes `dlrc-node1` to `dlrc-node5`. Each node has 2 cores and 4GB ram, but four SGE jobs are allowed to run simultaneously (only because otherwise there would be too little slots; the slot number may be even increased later).

Running Browser with Proxy to the Cluster

To run a browser, which can access master *and* the cluster nodes, run:

```
(chromium --proxy-server=socks://localhost:2020& \
ssh -ND 2020 -p 11422 ufallab.ms.mff.cuni.cz)
```

Wikipedia Links Data

The links between Wikipedia pages are available in the following directories:

- `/dlrc_share/data/wiki-links/cs-txt`: Czech Wikipedia pages links, 9.6M links, 500k pages, 328MB.
- `/dlrc_share/data/wiki-links/en-txt`: English Wikipedia pages links, 143.8M links, 11.8M pages, 5.6GB.

The files are encoded in UTF-8 and every line contain two space separated page names – the source page and the target page.

Tasks

When needing to split given text into words (called *tokens*), use function `wordpunct_tokenize` from `nltk.tokenize` package.

Task	*Points*	*Description*
`link_path`	4	For two given Wikipedia pages, find out the shortest path of page links between them, if it exists.
`transitive_closure`	4	Compute transitive closure of the Wikipedia link graph. In other words, compute for each page all pages reachable from it.
`page_rank`	5	Compute PageRank of Wikipedia pages and output the pages sorted by the rank. Use given number of iterations of the iterative algorithm `http://en.wikipedia.org/wiki/PageRank#Iterative` with damping factor 0.85.

K-Means algorithm

kmeans.py

```
#!/usr/bin/python
import sys
import numpy as np

from pyspark import SparkContext

def parseVector(line):
    return np.array([float(x) for x in line.split('␣')])
def closestPoint(p, centers):
    best = 0
    best_distance = np.sum((p - centers[0]) ** 2)
    for i in range(1, len(centers)):
        distance = np.sum((p - centers[i]) ** 2)
        if distance < best_distance:
            best, best_distance = i, distance
    return best
def kmeans(sc, file, clusters, epsilon):
    lines = sc.textFile(file, 3 * sc.defaultParallelism)
    data = lines.map(parseVector).cache()
    centers = data.takeSample(False, clusters, 1)
    distance = epsilon
    while distance >= epsilon:
        closest = data.map(lambda p: (closestPoint(p, centers), (p, 1)))
        coords = closest.reduceByKey(lambda s, t: (s[0] + t[0], s[1] + t[1]))
        newPoints = coords.map(lambda s: (s[0], s[1][0] / s[1][1])).collect()
        distance = sum(np.sum((centers[i] - p) ** 2) for (i, p) in newPoints)
        for (i, p) in newPoints:
            centers[i] = p
    print("Final␣centers:␣" + str(centers))

if __name__ == "__main__":
    if len(sys.argv) < 4:
        sys.stderr.write("Usage:␣kmeans␣input_file␣clusters␣epsilon\n")
        sys.exit(1)

    sc = SparkContext(appName="KMeans")
    kmeans(sc, sys.argv[1], int(sys.argv[2]), float(sys.argv[3]))
    sc.stop()
```

The example is available in `/home/straka/examples`.