Fast Convergence PageRank in Hadoop

Team Members:
Karthik Bellur(kvb26)
Sumukh SP(ss3345)
Prashanth Basappa(pb476)

Problem

Compute PageRank for a large scale web graph (685230 nodes and 7600595 edges) using a fast convergence using Blocked PageRanks

Idea is to reduce the I/O cost of a MapReduce job by doing nontrivial computation in the reduce step and increase the performance by reducing the number of passes required to achieve convergence.

Input Data

We pre-process the raw input data from edges.txt to create filtered_edges.txt.

We then perform some additional processing on the filtered_edges.txt file in order to bring it to a format which is convenient for the Mapper process to read.

```
double fromNetID = 0.5433;                        // using netid ss3345
double rejectMin = 0.9 * fromNetID;               // =0.48897
double rejectLimit = rejectMin + 0.01;            // =0.49897

boolean selectInputLine(double x) {
   return ( ((x >= rejectMin) && (x < rejectLimit)) ? false : true );
}
```

Mapper Input File Format:

<NodeID PageRank n D1 D2 D3 ….. Dn>

NodeID = Node ID of a source node
PageRank = Page Rank of the source node
D1 D2 D3 … Dn = List of destination nodes

Node-by-node Computation of PageRank

SimplePageRankMaster

- Bootstrapper for the MapReduce process
- Set up the Mapper & Reducer jobs; configure the input/output paths
- Create a MapReduce counter to store “residual”
- Output the residual (so we can conclude that PageRank is still far from convergence)

SimplePageRankMapper

Input:

<NodeID PageRank n D1 D2 D3 ….. Dn>

NodeID = Node ID of a source node

PageRank (PR) = Page Rank of the source node

D1 D2 D3 … Dn = List of destination nodes

n = Total number of nodes in the graph

Output:

```
For each node in <D1, D2, D3, … Dn>
    emit( NodeID, PageRank/n )
For each node in the graph
    emit( NodeID, “PR_” + <NodeID PageRank n D1 D2 D3 ….. Dn> )
```

Node-by-node Computation of PageRank

SimplePageRankReducer:

Input:

<NodeID, Inbound PageRank>

<NodeID, “PR_” + <NodeID PageRank n D1 D2 D3 ….. Dn>>

Output:

<NodeID, <NodeID NewPageRank n D1 D2 D3 ….. Dn>>

NewPageRank (NPR) = (1-d) / N + Inbound PageRank * d

The residual PageRank is calculated by the Reducer as shown below and added to a MapReduce counter.

Residual PageRank = Math.abs(NewPageRank - PageRank) / NewPageRank

The average residual PageRank over all the nodes in the graph is calculated as shown below:

Avg Residual PageRank = Residual PageRank / n;

Results

```
Iteration 1, Avg Error: 2.3390409059731767
Iteration 2, Avg Error: 0.3228536403835209
Iteration 3, Avg Error: 0.19199830713774937
Iteration 4, Avg Error: 0.09401514819841512
Iteration 5, Avg Error: 0.06275703048611415
Iteration 6, Avg Error: 0.03391270084497176
Iteration 7, Avg Error: 0.027217138770923632
Iteration 8, Avg Error: 0.0165739970520847
Iteration 9, Avg Error: 0.01436597930621835
Iteration 10, Avg Error: 0.009770442041358376
Iteration 11, Avg Error: 0.008454095705091721
Iteration 12, Avg Error: 0.006145381842592998
Iteration 13, Avg Error: 0.005370459553726486
Iteration 14, Avg Error: 0.00402346657326737
Iteration 15, Avg Error: 0.0035068517140230287
Iteration 16, Avg Error: 0.002715876421055704
Iteration 17, Avg Error: 0.0023408198707003487
Iteration 18, Avg Error: 0.0018533922916393036
Iteration 19, Avg Error: 0.0015907067699896386
Iteration 20, Avg Error: 0.0012754841440100403
Iteration 21, Avg Error: 0.00109014491484611
Iteration 22, Avg Error: 8.858339535630373E-4
```

Blocked Computation of PageRank

Node to Block Mapping:
The MapReduce master process, on startup, reads from the blocks.txt file and creates a list consisting of the prefix sums of the values contained in the blocks.txt file. A binary search of this list would then give us the corresponding Block for the Node under consideration in O(log(numberOfBlocks)) time.

BlockedPageRankMapper
Input:

    <NodeID PageRank n D1 D2 D3 ….. Dn>

    NodeID = Node ID of a source node
    PageRank (PR) = Page Rank of the source node
    D1 D2 D3 … Dn = List of destination nodes
    n = Total number of nodes in the graph

Blocked Computation of PageRank

Output:

```
For each node in the graph
    emit( BlockID, “PR ” + <NodeID PageRank n D1 D2 D3 ….. Dn> )
For each node in <D1, D2, D3, … Dn>
    If( blockIdOfNode( Di ) == blockIdOfNode( NodeID ) )
        emit( blockIdOfNode( Di ), “BE ” + NodeID + “ ” + Di )
    else
        emit( blockIdOfNode( Di ), “BC ” + NodeID + “ ” + Di + “ ” + PR / n )
```

Blocked Computation of PageRank

```
BlockedPageRankReducer:
do until convergence:
    For each Value for block b
        BE <- Set of edges in the same block b
        BC <- Set of incoming edges to block b

        For each node v in block b
            For each node u: edge(u,v) is in b
                sum incoming page rank for v
            For node u : edge(u,v) is an incoming edge for a different block b'
                sum incoming page rank for v
            newPageRank(v) = (1-d)/n + d*(sum of incoming page ranks to v)
```

Residual PageRank = Math.abs(NewPageRank - PageRank) / NewPageRank

emit (v , new PageRank(v), n , D1,D2,....Dn)

Convergence Condition:

```
    Average Residuals / N <= 0.01
```

Results

```
Iteration 1, Avg Error: 2.815142360959094, Avg iterations per block = 17.58823529411765
Iteration 2, Avg Error: 0.03704449600863943, Avg iterations per block = 7.132352941176471
Iteration 3, Avg Error: 0.02345781708331509, Avg iterations per block = 5.955882352941177
Iteration 4, Avg Error: 0.009277177006260672, Avg iterations per block = 3.911764705882353
Iteration 5, Avg Error: 0.003460152065729755, Avg iterations per block = 2.411764705882353
Iteration 6, Avg Error: 0.001113494738992747, Avg iterations per block = 1.5294117647058822
Iteration 7, Avg Error: 6.041766997942297E-4, Avg iterations per block = 1.1323529411764706
```

Gauss Siedel PageRank

Nodes in the graph are sorted.

New PageRank values are used in successive computation of PageRank values in the same iteration.

Results:

Iteration 1, Avg Error: 2.815524714329495, Avg iterations per block = 10.088235294117647

Iteration 2, Avg Error: 0.038112750463348076, Avg iterations per block = 5.073529411764706

Iteration 3, Avg Error: 0.02474789486741678, Avg iterations per block = 4.470588235294118

Iteration 4, Avg Error: 0.010522014506078252, Avg iterations per block = 3.2205882352941178

Iteration 5, Avg Error: 0.004513812880346745, Avg iterations per block = 2.3676470588235294

Iteration 6, Avg Error: 0.001625731506209593, Avg iterations per block = 1.6176470588235294

Iteration 7, Avg Error: 7.413569166557215E-4, Avg iterations per block = 1.2647058823529411

Random Blocked PageRank Computation

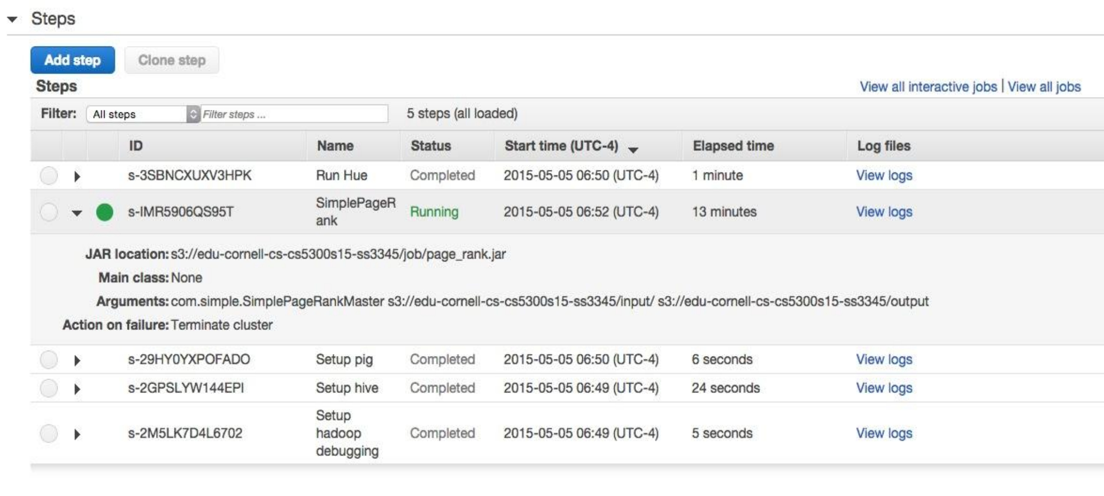
Randomly assign nodes to blocks instead of using the METIS graph partition

We used a simple function: (nodeId * 541) % 68;

Results:

```
Iteration 1, Avg Error: 2.33966405440509
Iteration 2, Avg Error: 0.3223165944281482
Iteration 3, Avg Error: 0.19119565693270874
Iteration 4, Avg Error: 0.0934357806867767
Iteration 5, Avg Error: 0.06209010113392584
Iteration 6, Avg Error: 0.0335172131984881
Iteration 7, Avg Error: 0.02690045678093487
Iteration 8, Avg Error: 0.016401792099003255
Iteration 9, Avg Error: 0.014182099441063585
Iteration 10, Avg Error: 0.009666827196707676
```

EMR Deployment



Steps

Add step | Clone step

Steps

View all interactive jobs | View all jobs

Filter: All steps | Filter steps ... | 5 steps (all loaded)

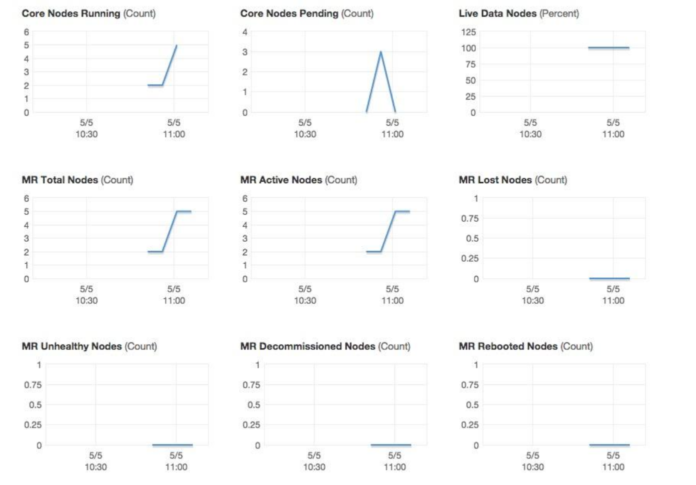
ID	Name	Status	Start time (UTC-4)	Elapsed time	Log files
s-3SBNCXUXV3HPK	Run Hue	Completed	2015-05-05 06:50 (UTC-4)	1 minute	View logs
s-IMR5906QS95T	SimplePageRank	Running	2015-05-05 06:52 (UTC-4)	13 minutes	View logs
s-29HY0YXPOFADO	Setup pig	Completed	2015-05-05 06:50 (UTC-4)	6 seconds	View logs
s-2GPSLYW144EPI	Setup hive	Completed	2015-05-05 06:49 (UTC-4)	24 seconds	View logs
s-2M5LK7D4L6702	Setup hadoop debugging	Completed	2015-05-05 06:49 (UTC-4)	5 seconds	View logs

JAR location: s3://edu-cornell-cs-cs5300s15-ss3345/job/page_rank.jar

Main class: None

Arguments: com.simple.SimplePageRankMaster s3://edu-cornell-cs-cs5300s15-ss3345/input/ s3://edu-cornell-cs-cs5300s15-ss3345/output

Action on failure: Terminate cluster



Thank you!