HITS Algorithm on Twitter Graph

By Nikhil Kathuria and Kishore Sridhar

Twitter Dataset - 2 Billion Edges

- Publicly available dataset containing a snapshot of Twitter's followers graph from September 2009.
- The graph is stored as a single text file having 1, 963, 263, 821 (approx 2 billion) edges.
- The size of the decompressed dataset is 38.8 GB in plain text format.
- From the edge representation, we found 52, 579, 682 (52.5 million) unique nodes, i.e. number of users which followed or was followed by at least 1 other user.

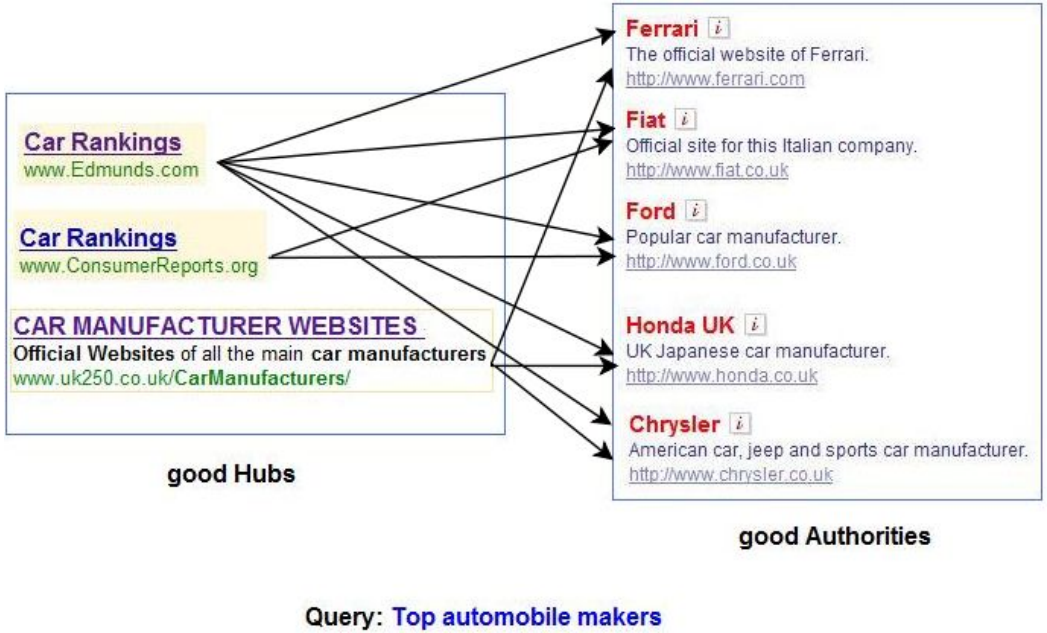
Objective : First-login recommendation system

- We look to provide *'suggested users'* for brand new users to follow on a social network during first login.
  - Since they are brand new, we do not have any data about them to aid our search.
  - A solution that is based purely on the followers graph would be ideal.
  - Most followed users may not always be good authorities.
  - A PageRank-like approach is often more useful, i.e. the credibility of a node depends on the credibility of those who point to it.

Approach : HITS Algorithm

- Hyperlink Induced Topic Search (HITS) is an algorithm that finds good hubs and good authorities in a given graph.
- Good Authority - Contains/provides authoritative content on a given query and is pointed to by several good hubs
- Good Hub - Points to several good Authorities.
- An iterative algorithm that operates on a small subgraph of the original graph which is usually obtained using a query.
  - We obtain this subgraph based on the following assumption.
- Assumption: The potential auths lie in the set of users which satisfy both:
  - Is followed by the users who follow the highest number of users
  - Has at least k followers (our k = 1500)

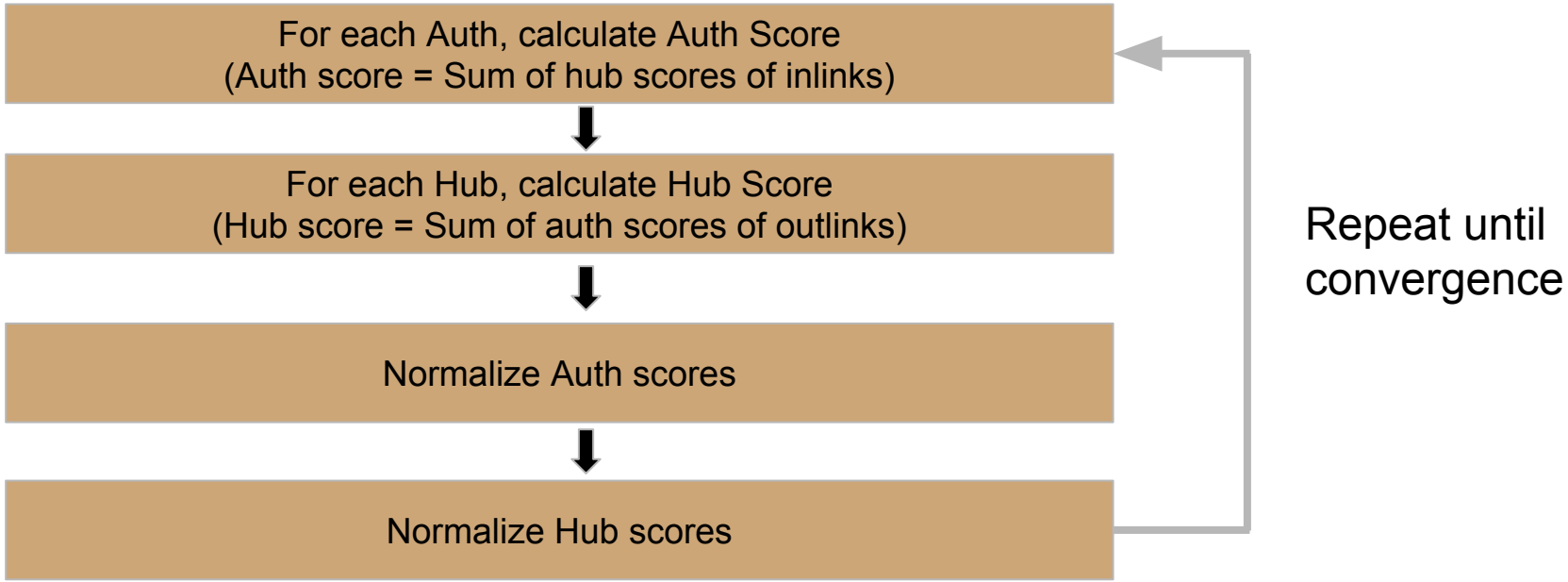
Authorities and Hubs



Query: Top automobile makers

The best authorities would be the suggestions for the new users.

HITS : Each Iteration

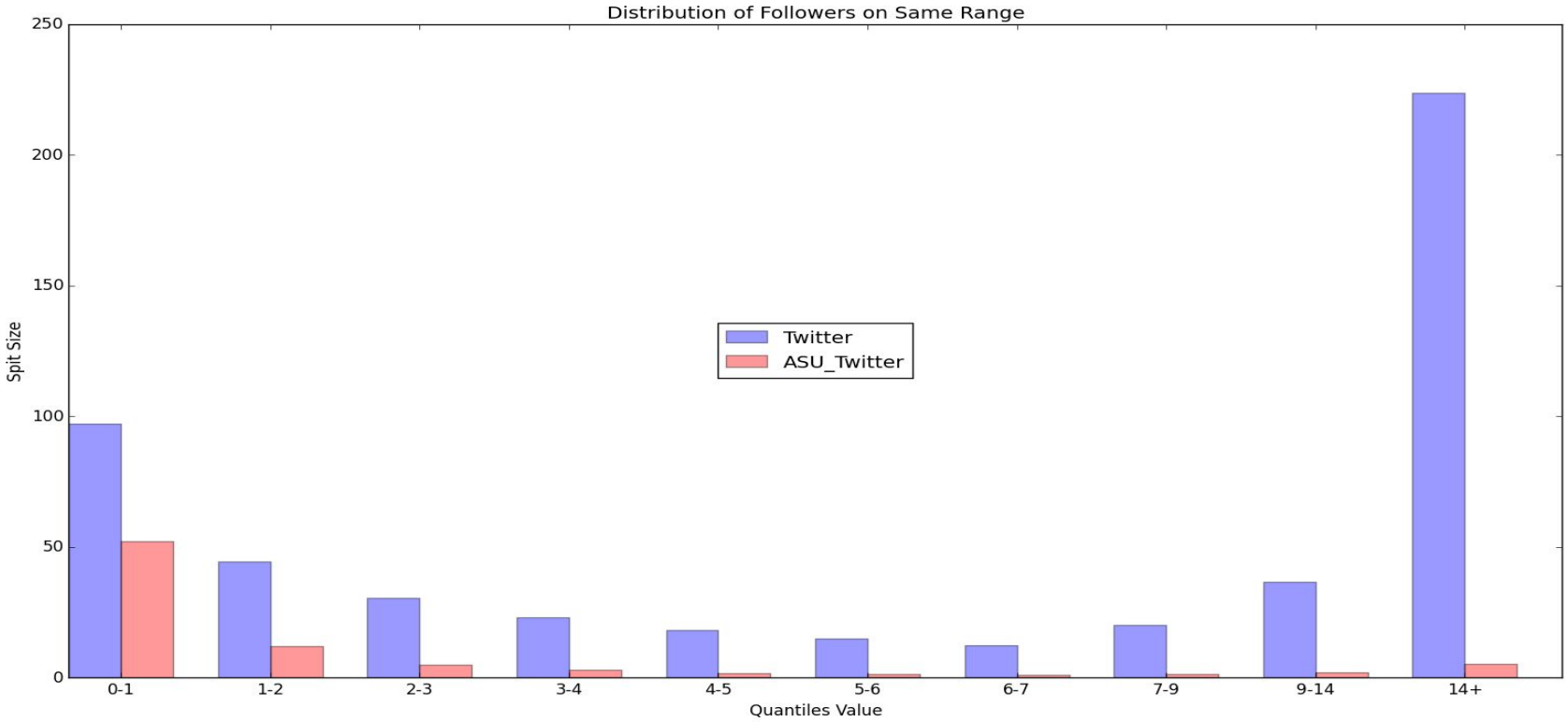


Running an iterative algorithm on MapReduce

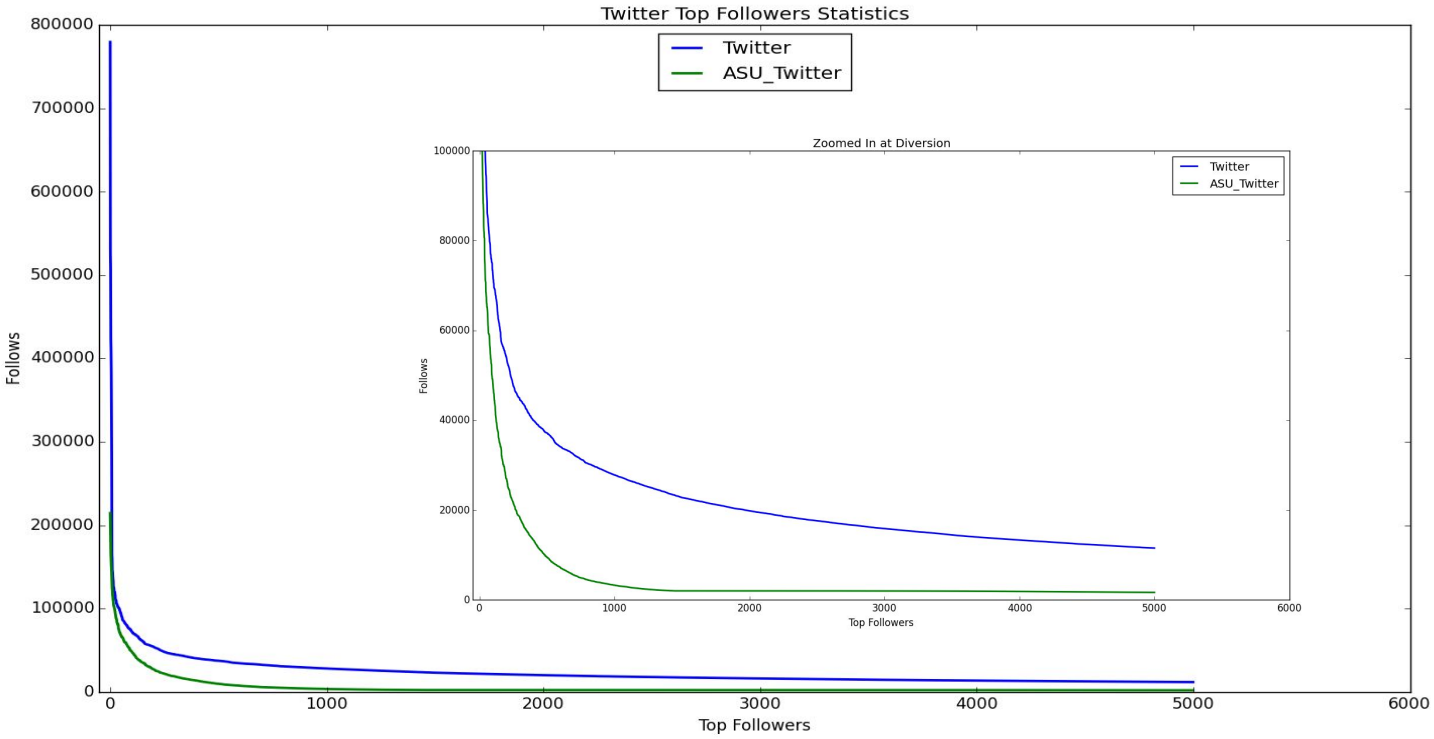
```
public static void main(String[] args)
{
        while(break_condition)
        {
                conf = new Configuration();
                .
                .
                .
                job.waitForCompletion(true);
                count++;
                args = updateArgs(args, count);
        }
}
```

```
updateArgs(args, Iteration_count)
{
String outDir = parent + (Iteration_count + 1) + "/" ;
String inDir = parent + count + "/";
String inputFile = inDir + part;

// Generate Generic parser options
for (int itr =1; itr < genOp.length; itr++){
   string = string + "," + genOp[itr];
}

// Update the output, input directory and Generic options
args[args.length - 1 ] = outDir;
args[args.length - 2] = inDir;
args[1] = string;
}
```

Highlights : Distributions on two datasets



Highlights: Top followers statistics



Highlights : Parallelizing Map by Altering Split Size

- Reduce tasks can be parallelized explicitly by using the Job. setNumReduceTasks() method.
- The number of Map tasks that are created for a given job depends on the number of splits of the input file(s).
- The split size can be altered in the configuration.

```
Configuration conf = new Configuration();
long MAXSIZE =  3551097;  // file size: 53236459, for 15 workers
conf.setLong(FileInputFormat.SPLIT_MAXSIZE, MAXSIZE);
```

Highlights : Custom Partitioner

- Custom Range Order Partitioner
  - In order to sort the outlinks based on number of outlinks, we implemented our own range order partitioner using quantiles that we found from a smaller alternative Twitter dataset.

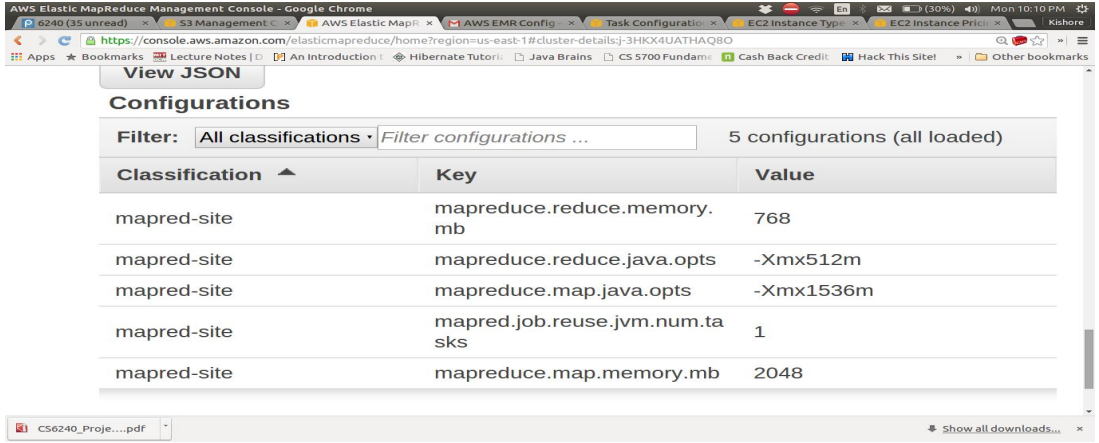
All Buckets / kish.project.output / out_descsorted

Name	Storage Class	Size	Last Modified
_SUCCESS	Standard	0 bytes	Thu Dec 03 22:58:10 GMT-500 2015
part-r-00000	Standard	97 MB	Thu Dec 03 22:57:35 GMT-500 2015
part-r-00001	Standard	44.4 MB	Thu Dec 03 22:57:43 GMT-500 2015
part-r-00002	Standard	30.3 MB	Thu Dec 03 22:57:45 GMT-500 2015
part-r-00003	Standard	23.1 MB	Thu Dec 03 22:57:48 GMT-500 2015
part-r-00004	Standard	18.3 MB	Thu Dec 03 22:57:51 GMT-500 2015
part-r-00005	Standard	14.9 MB	Thu Dec 03 22:57:54 GMT-500 2015
part-r-00006	Standard	12.4 MB	Thu Dec 03 22:57:57 GMT-500 2015
part-r-00007	Standard	20.2 MB	Thu Dec 03 22:57:59 GMT-500 2015
part-r-00008	Standard	36.6 MB	Thu Dec 03 22:58:02 GMT-500 2015
part-r-00009	Standard	223.8 MB	Thu Dec 03 22:58:07 GMT-500 2015

```
kishore@UbuntuBook: ~/Documents/mapreduce/project/asu/descsorted2
kishore@UbuntuBook:~/Documents/mapreduce/project/asu/descsorte
total 84M
-rw-r--r-- 1 kishore kishore  52M Dec  3 22:12 part-r-00000
-rw-r--r-- 1 kishore kishore  12M Dec  3 22:13 part-r-00001
-rw-r--r-- 1 kishore kishore 5.0M Dec  3 22:13 part-r-00002
-rw-r--r-- 1 kishore kishore 2.8M Dec  3 22:13 part-r-00003
-rw-r--r-- 1 kishore kishore 1.8M Dec  3 22:13 part-r-00004
-rw-r--r-- 1 kishore kishore 1.3M Dec  3 22:13 part-r-00005
-rw-r--r-- 1 kishore kishore 939K Dec  3 22:13 part-r-00006
-rw-r--r-- 1 kishore kishore 1.3M Dec  3 22:13 part-r-00007
-rw-r--r-- 1 kishore kishore 2.0M Dec  3 22:13 part-r-00008
-rw-r--r-- 1 kishore kishore 5.2M Dec  3 22:13 part-r-00009
-rw-r--r-- 1 kishore kishore    0 Dec  3 22:13 _SUCCESS
-rw-rw-r-- 1 kishore kishore  60K Dec  7 19:58 tmp_asu.txt
kishore@UbuntuBook:~/Documents/mapreduce/project/asu/descsorte
```

Highlights : EMR Configurations

- Custom configuration in EMR
  - Used a custom configuration in EMR to increase the memory and heap space assigned to map and reduce tasks.
  - Enabled us to run a job that required to use about 1400 MB on m1.medium instances by increasing the task and heap allocation values



Highlights: Multiple Mapper Classes in a Job

The requirement to merge the output of two independent MapReduce jobs called for two mapper classes, whose output was merged by the same reducer class.

```
99    Job job = new Job(conf, "MergeLinks");
100   job.setJarByClass(MergeLinks.class);
101
102   MultipleInputs.addInputPath(job, new Path(otherArgs[0]), TextInputFormat.class, MapOut.class);
103   MultipleInputs.addInputPath(job, new Path(otherArgs[1]), TextInputFormat.class, MapIn.class);
104   job.setReducerClass(MAReduce.class);
105   job.setOutputKeyClass(Text.class);
106   job.setOutputValueClass(Text.class);
```

S.No	Name	Input Size	Output Size	Run Time (mins)	EMR Machines
1	Generate Inlinks	38.8 GB	16.2 GB	56	1+10 (m1.medium)
2	Generate Outlinks	38.8 GB	18 GB	58	1+10 (m1.medium)
3	Sort based on outlink count	18 GB	1 GB	18	1+10 (m1.medium)
4	Build Authset	18 GB	100 KB	15	1+10 (m1.medium)
5	Filter inlinks based on authset	16.2 GB	1.4 GB	26	1+10 (m1.medium)
6	Trim Outlinks	18 GB	14 GB	30	1+10 (m1.medium)
7	Hub Expansion	14 GB	29 MB	150	1+10 (m1.medium)
8	Filter outlinks based on exp hub	14 GB	11 GB	28	1+10 (m1.medium)
9	Outlinks needed for HITS	11 GB	1.3 GB	15	1+10 (m1.medium)
10	Inlinks needed for HITS	1.4 GB	1.3 GB	4	1+10 (m1.medium)
11	Merge inl and outl to one file	2.6 GB	2.6 GB	14	1+2 (m1.medium)
12	Merge hub and auth to one file	29.1 MB	50.7 MB	3	1+2 (m1.medium)
13	HITS Algorithm	2.6 GB	2.8 GB	190	1+15 (m3.xlarge)

References

http://stanford.edu/~rezab/papers/wtf_overview.pdf

https://www.cs.cornell.edu/home/kleinber/auth.pdf

Hadoop: The Definitive Guide, 4th Edition

http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html

Thank You