CS 677: Big Data

Hadoop MapReduce

Lecture 13

Project 2

- The spec for Project 2 is up!
- We will build our own MapReduce implementations to go with Project 1
- Today we'll look at Hadoop's implementation of the MapReduce paradigm
  - What better way to come up with our design than to look at the competition?

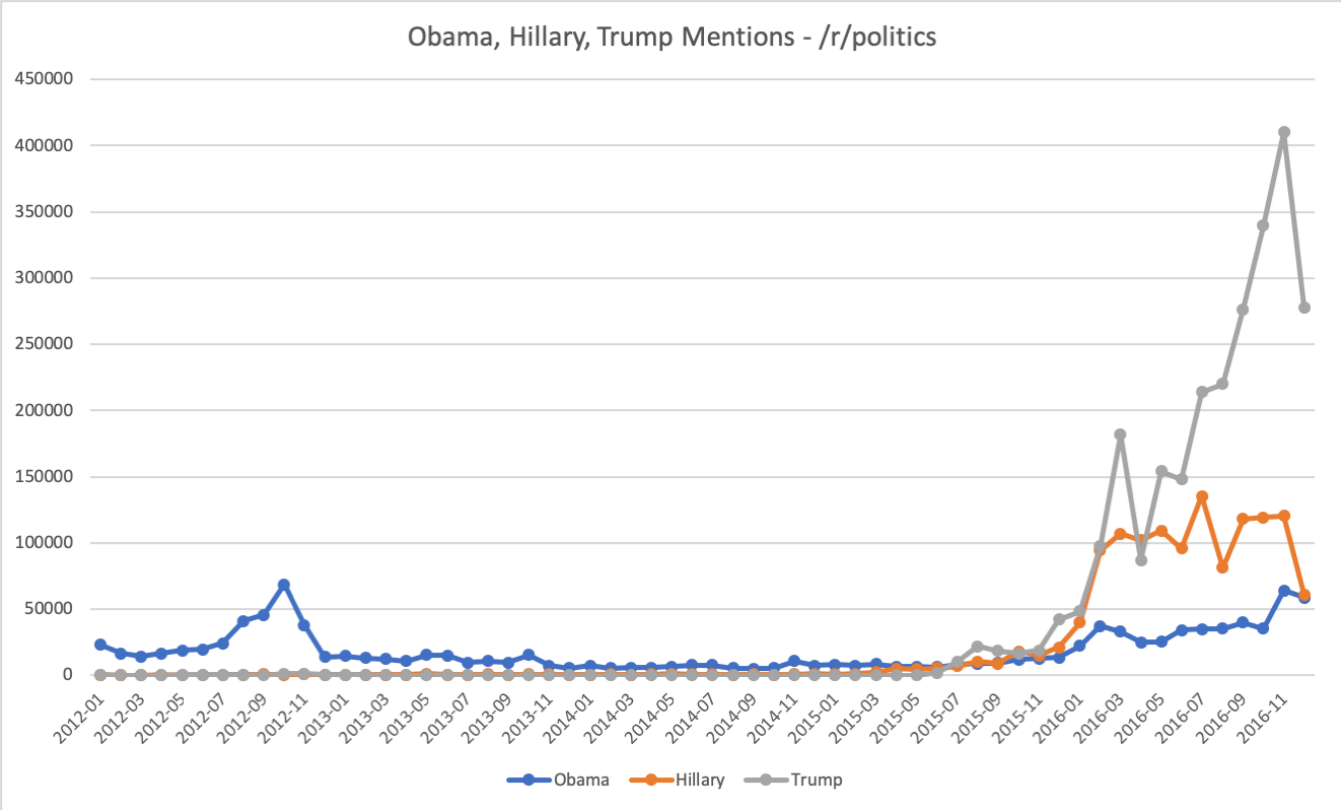
Motivation: Social Media

- You can gain a lot of insight about what's going on in the world with social media
  - A deep, dark cesspit of sorrow!
- Good places to mine for information and influence elections: Twitter, TikTok, Reddit, Facebook … etc?
- Let's look at one example using Reddit comments

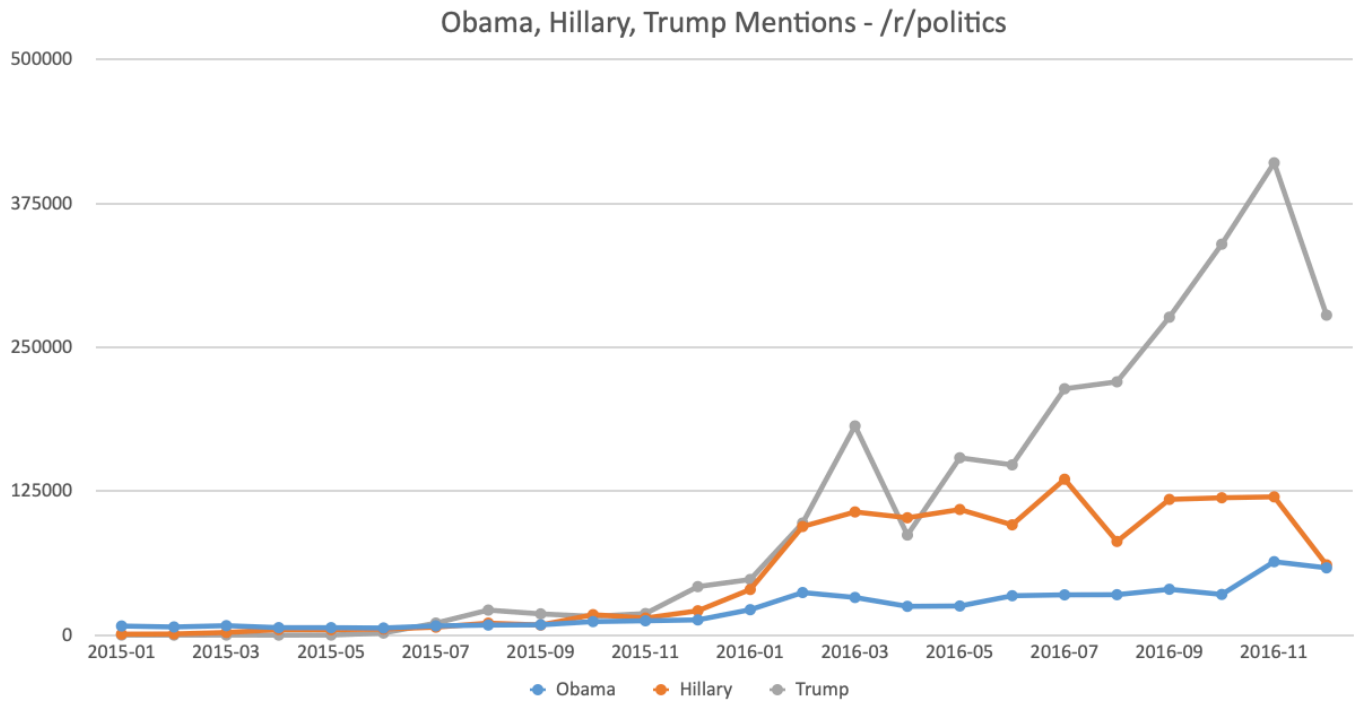
2016 US Presidential Election

- Reddit is broken into a huge number of communities called *subreddits*
- As an example, let's scan through posts under `/r/politics`
  - In 2016
- Every time we see a mention of Obama, Trump, or Hillary, we'll increment a counter
- Question: What does this look like over time?

2012 – 2016


Obama, Hillary, Trump Mentions - /r/politics
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
2012-01
2012-03
2012-05
2012-07
2012-09
2012-11
2013-01
2013-03
2013-05
2013-07
2013-09
2013-11
2014-01
2014-03
2014-05
2014-07
2014-09
2014-11
2015-01
2015-03
2015-05
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
Obama
Hillary
Trump

2015 – 2016


Obama, Hillary, Trump Mentions - /r/politics
500000
375000
250000
125000
0
2015-01
2015-03
2015-05
2015-07
2015-09
2015-11
2016-01
2016-03
2016-05
2016-07
2016-09
2016-11
Obama
Hillary
Trump

Today's Schedule

- Hadoop MapReduce Components
- Application Scheduling
- Hadoop Tips

Today's Schedule

- **Hadoop MapReduce Components**
- Application Scheduling
- Hadoop Tips

MapReduce: The Hadoop Flavor

- Last week, we covered the general MapReduce programming paradigm
- Let's dig a little deeper into the most popular implementation of MapReduce: Hadoop!
  - (So we can steal their good ideas for P2)

Components

- We've studied the various parts of HDFS already…
  - NameNode, DataNode, SecondaryNameNode
- What about the compute side of things?
- "MapReduce 2.0," a.k.a. **YARN** (Yet Another Resource Negotiator)

  NodeManager
  - ResourceManager
  - ApplicationManager
  - NodeManager

NodeManager

- A NodeManager instance runs on each compute node in the cluster
- Receives instructions from the ResourceManager on what to run
  - Includes a copy of your application .jar
- Each YARN application is run inside a *container* on the NodeManager
  - Note: no relation to Linux containers / Docker / etc.
  - More like an isolated JVM instance

ResourceManager

- One ResourceManager is required to operate a YARN cluster
  - You'll set the hostname/port of the ResourceManager in the Hadoop config
  - **Also** must start the ResourceManager from the correct host!
    - Not the case for NodeManagers, DataNodes, etc.
- Handles cluster management, assigning tasks, scheduling, etc.

RM Responsibilities

- Maintaining the list of NodeManagers (compute nodes)
- Client/admin RPC functionality
- Liveliness monitor: tracks NM heartbeats
- Access control, permissions, security
- Task scheduling

ApplicationManager

- When you run a YARN job, it is managed by… you guessed it, the ApplicationManager!
- The AM runs inside a container on one of the compute nodes
- Tracks job status, monitors execution, reports the results back to you

HDFS + Yarn

- HDFS and Yarn are decoupled
- If you change the HDFS config, you only need to restart it (not yarn!)
- In fact, you can run Yarn + MR on a different distributed file system if you want
- Spark, another distributed computation engine, can run on YARN
  - Several systems can do this, in fact!

Today's Schedule

- Hadoop MapReduce Components
- **Application Scheduling**
- Hadoop Tips

Scheduling Algorithms

- The ResourceManager supports three scheduling algorithms:
  1. FIFO
  2. Fair Scheduling
  3. Capacity Scheduling
- What scheduler you use depends on your organization's needs
  - If you are the only user of the cluster, then FIFO usually works fine 😄

Fair Scheduling

- Goal: on average, applications/jobs should be given an equal share of resources
- If Job A is running and Job B is submitted, the tasks for Job B will start running once some of Job A's tasks finish
  - Tasks are allocated by rotating through pending applications
  - Everyone gets a fair share (over time)
- By default, the fair scheduler bases its scheduling decisions on memory usage
  - Can be configured to use CPU as well

Capacity Scheduling [1/2]

- Organizations generally run Hadoop clusters that can operate near their peak required capacity
  - Especially important if providing service-level agreements (SLAs)
- This deployment strategy is inefficient when the cluster is not always running at maximum capacity
  - …but we don't want to share it with anyone because then resources might be tied up when we need them!

Capacity Scheduling [2/2]

- The capacity scheduler enables organizations to have a shared cluster with **resource guarantees**
- Each org gets a fraction of the cluster resources, e.g., 30%
  - If nobody is using the other available resources, then your application can use them as well!
- However, both hard and soft limits can be applied
  - Perhaps you want to ensure 10% head room is always available for a certain org, or disallow orgs to go beyond their allocated share

Priorities

- Applications can also have priorities assigned to allow finer-grained control over what runs first
- Applies to all three algorithms:
  - FIFO: higher priority jobs will run first
  - Fair: priority determines the weight of the applications, increasing their share of the resources
  - Capacity: within a queue, higher priority applications are run first
    - Note: does not impact other organizations

Speculative Execution

- Rather than trying to figure out why a task has failed, Hadoop launches **speculative tasks**
  - Also known as **backup tasks** (Google terminology)
- These are task duplicates that are scheduled to run on different machines than their original copy
  - The other machine might be overloaded, may just be slower than average, etc.
  - Which result do you keep? The one that finishes first!
- For some large jobs, Google found it took 44% longer to finish without speculative execution. Only a small percentage of duplicates are launched for each job

Today's Schedule

- Hadoop MapReduce Components
- Application Scheduling
- **Hadoop Tips**

Rebalancing

- The block placement strategy in HDFS is unfortunately not always very smart
  - It makes some assumptions that may hurt performance in our case
- Run: `hdfs balancer -threshold 1`
  - You can increase the bandwidth for this, but probably should just let it run for a while

Operating on Local Files

- Hadoop is a little weird: it operates on local files by default if you haven't configured HDFS yet
- Once HDFS is set up, it assumes the files are coming from there
- You can still refer to local files, though:
  - Specify `file:///home/username/file` as an input or output to use non-HDFS paths

Cleaning Up

- If you have a hung job blocking your run queue, you will need to kill it
- To do this: `yarn application –kill <app_id>`
- App IDs are shown in `yarn top` or near the top of your job output

Being Lazy

You can use the LazyOutputFormat to avoid writing empty files during the reduce phase

```
import org.apache.hadoop.mapreduce.lib.output.LazyOutputFormat;

. . .

LazyOutputFormat.setOutputFormatClass(job, TextOutputFormat.class);
```

Cleanup() Method

Let’s assume you populate a HashMap with values during the map phase. You can then emit a condensed version during cleanup:

```
@Override
protected void cleanup(Context context)
throws IOException, InterruptedException {
    for (Text geohash : hottest.keySet()) {
        Double temp = hottest.get(geohash);
        context.write(geohash, new DoubleWritable(temp));
    }
}
```

Setup() Method

- There is also a setup() method you can override
- Not as useful as cleanup(), but can be used to initialize things before the task begins

Hrm…

- Abusing the `setup()` and `cleanup()` methods (especially `cleanup()`) tends to circumvent the framework
- Example: I'm going to stick everything in a hashmap, and then emit a bunch of k,v pairs at the end
  - This can work… but MR already is basically doing this for you
  - Plus: what happens if you can't fit the hashmap in memory?

Custom Writables

- You don’t have to re-encode output data as text or JSON
- If you are going to emit multiple values, encapsulate them in a custom writable
  - Public members are totally fine. No need for fancy constructors, etc.
- This will improve performance and reduce the amount of duplicate work you do

Custom Input Formats

- You aren't stuck with simple `<line_no, text>` KV pairs
- You can design your own input format, or extend an existing one
- For example: NLineInputFormat when you want to read multiple lines at a time

Custom Output Formats

- You can also write your own output formats
- Not too much work – implement some methods
- Here's how you can write your own format that doesn't produce empty files: http://whiteycode.blogspot.it/2012/06/hadoop-removing-empty-output-files.html