# CSCI 339 Project 3: Contextual Advertising

Erik Kessler, Kevin Persons
*Williams College*

## Abstract

We describe the setup, architecture, and implementation details of our Hadoop cluster and Hadoop jobs that use the power of distributed computing and the MapReduce framework to process impression and click logs in order to compute the click-through rates for advertisements on various websites. In this paper, we also detail our extensions to the project and conclude with our general thoughts and reflections on the project and the Hadoop framework.

## 1 Introduction

With this project we aimed to gain experience with Hadoop. Hadoop is an open-source implementation of Google's MapReduce data-processing model. The framework relies on a simple model drawn from functional programming languages of mapping and reducing. The functional and simple nature of the model makes it possible to parallelize the computation on a distributed cluster of machines allowing fast processing of extremely large data-sets.

The problem presented to us in this project was we were given a large number of log entries for advertisement impressions and advertisement clicks. Our task was to process those log entries to compute the click-through rate, which is the rate that users actually click on an ad, for each website, ad pair. This information would be useful if we were looking to determine which advertisements are working well and should be displayed on a given page. Given the size of the dataset, running any computation over the data on a single machine would take a while, so Hadoop provides a promising way to speed up the computation by distributing the work over multiple machines. In this paper we discuss our cluster setup, the architecture of the Hadoop jobs, extension, along with some general reflections on the project and framework.

## 2 Architectural Overview

The first step in our project involved setting up a cluster to run Hadoop. In order to gain experience with current technologies and set up a cluster quickly and cheaply we used Amazons Elastic Compute Cloud (EC2) infrastructure which allowed us to create free virtual machines to run our jobs. Our EC2 setup included four t2.micro Ubuntu virtual machine instances which means each machine had 1 virtual CPU, 1GB of memory, 8GB of storage. As the free-tier, these machines are relatively low-powered, but were powerful enough for this project. We had to distribute and set up Hadoop on each machine. We then assign one machine to be the NameNode, meaning it acts as the master, another to be the secondary NameNode, and two to be slave nodes. Lastly, Hadoop uses a distributed file system, so we had to load our data set onto the Hadoop File System.

Now we describe the design of our Hadoop jobs, namely we describe the map and reduce functions we wrote. The solution we used involved chaining two jobs together because we had to do a single map and two reduces. The main problem is

that click log entries do not have the page attribute while the impression entries do not know if the impression was clicked or not. As a result, the goal of the first job was to get a more complete picture for each impression: page, ad_id, and whether or not it was clicked. So our first mapping function `ImpressionMapper` reads the log entries and uses a JSON library to parse the entry. If the entry is an impression the mapper emits an entry with the impression ID as the key and [page, ad_id] as the value. If the entry is a click it emits an entry with the impression ID as the key and CLICKED as the value.
Next, the first reducer, `ImpressionReducer`, will get either: a single entry if the impression was not clicked so we emit [page, ad_id] as the key and 0 as the value to indicate no click, or two entries if the impression was clicked so we emit [page, ad_id] with the value of 1 to indicate a click. Now we have a bunch of single actions (impressions with no clicks and impressions with clicks) for each page, ad pair so we need to reduce all of those individual actions into a single click-rate value. The second reducer, `ClickRateReducer`, does this by counting the number of 1s and dividing that value by the total number of mappings. A sample of the output from each phase is in Appendix A.
There is a tradeoff to doing two jobs because we have to map and reduce twice. For one, the second map was just an identity map which is fast and the reduce was relatively simple which made the second job fairly fast. The benefit of using two jobs is that it keeps the mapping and reducing functions very simple. Maybe we could have done it in one job if we had mapped to have ad IDs be the key, but then we would have had to make our reduce function very complex and probably slow. Also our implementation makes it easier to extend the program because the first job just gathers a more complete picture of the action, so if we wanted to include, say the user-agent, we could fairly easily include that as a piece of the data that is associated with an impression id.

## 3 Evaluations and Extensions

Running our job on the four machine EC2 cluster with the full dataset took about 1 minute and 15 seconds which is impressive given we are working with a million impressions.

Hadoop offers a variety of ways to tune the system to extract better performance. One parameter we tested was the number of reduce tasks (using the -Dmapred.reduce.tasks=n flag). By default it runs with a single reduce task so we wanted to see how increasing that would impact performance. Interestingly, we found that increasing the number of reduce tasks slowed the process. When we included two reduce tasks, the time to process the dataset increased to about 1 minute and 56 seconds. Ramping up to five tasks, performance got even worse and the same task took 2 minutes and 12 seconds. With twenty tasks, the task ran for 4 minutes and 25 seconds.
Perhaps our cluster is not powerful enough to handle multiple tasks so splitting it up introduces extra overhead and work that slows it down, or perhaps there are other parameters we need to tweak as well. Possible extensions might involve seeing the impact of adding more machines to the cluster and tweaking other parameters of the system. Furthermore, the EC2 instances we are running are relatively low powered so it might be interesting to see how performance changes with more powerful machines.

The extension that we implemented is another Hadoop job that serves to find the best ad (highest click-through rate) for a given website. Again, the MapReduce model made writing this function relatively simple as we just had the map function emit the page url as the key and an ad id, click rate pair as the value. The reducer therefore receives all ads for a given page and can just loop through and find the one with the highest click-through rate.

## 4 Thoughts and Conclusions

In this project we gained experience with building virtual clusters in EC2 and with using the Hadoop framework. It was interesting to work within a distributed platform like Hadoop because it showed both the great benefits of abstracting the complexities of the underlying distributed cluster

as well as the difficulties. On the one hand, while writing our MapReduce task we didn't need to worry about how to parallelize the computations being done or managing the virtual machines. This transparency made creating what we wanted pretty straightforward and easy once we understood the framework. On the other hand, however, Hadoop can be hard to debug and tune because of these abstractions. It is hard to see what the system is actually doing which, as we've seen, is a tradeoff.

Overall we enjoyed this assignment. It was beneficial to work with EC2 and to work with the MapReduce framework to get some experience thinking in that way. The hardest part of the assignment was figuring out some of the specifics of Hadoop like setting up jobs and specifying file paths. It was helpful to have the examples so that we could see where to start, but it still took some time to go through and understand what each piece does. Once we had Hadoop configured and we could focus on solving the puzzle of how to efficiently make use of the MapReduce model, writing the code was fairly straightforward. This assignment didn't take very long to complete, we spent approximately seven hours including the initial setting up of EC2 during class time. We thought that this assignment was valuable due to the exposure to EC2 and the mindset of writing MapReduce tasks, so we would recommend it for future classes.

## A Sample Output

`ImpressionMapper` Output:

```
U2A3oETg5enessW7239C2xChDdAk2n    [21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]
medDGrfntwtBF5juF5b5jcmGmEH5BN    [21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]
VOjFxJt3s4IMe4N39QQStVnF8hixpj    [21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]
VOjFxJt3s4IMe4N39QQStVnF8hixpj    CLICKED
```

`ImpressionReducer` Output:

```
[21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]    0
[21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]    0
[21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]    1
[21cn.com, 0Gh5BouJWwnIWF2gSJSWnEV7xXOS1J]    0
[21cn.com, 0HnHLlcJWRBI58AftUWC5XoRRHvnwb]    1
[21cn.com, 0HnHLlcJWRBI58AftUWC5XoRRHvnwb]    1
[dmoz.org, N9aeSOorER3teEepOsHBepI4XQNiML]    0
[dmoz.org, N9aeSOorER3teEepOsHBepI4XQNiML]    0
[dmoz.org, N9aeSOorER3teEepOsHBepI4XQNiML]    1
[dmoz.org, N9aeSOorER3teEepOsHBepI4XQNiML]    0
[dmoz.org, NAAQgFa2kPVUsqjJRhLUmLN1I4BS2i]    1
[dmoz.org, NB7BK3OxrjawnaU1m94CTQnSdDQnw5]    0
```

`ClickRateReducer` (Final) Output:

```
[21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]    033
[21cn.com, 0Gh5BouJWwnIWF2gSJSWnEV7xXOS1J]    000
[21cn.com, 0HnHLlcJWRBI58AftUWC5XoRRHvnwb]    100
[dmoz.org, N9aeSOorER3teEepOsHBepI4XQNiML]    025
[dmoz.org, NAAQgFa2kPVUsqjJRhLUmLN1I4BS2i]    100
[dmoz.org, NB7BK3OxrjawnaU1m94CTQnSdDQnw5]    000
```

`RecommendedAd` Sample Output:

```
cbssports.com   00dAqnPbhpOSM9uDfqjVIQpqV3G4Fh
clubpenguin.com 09tkspRaCqqAKhMBTv2X8aDgG44pmu
corriere.it     0JF3P6kWLUtTWXmMUq0iwjTRX3x0Pg
csdn.net        06sJVbSkJNqXfEnxTRVrW6r9oqHpQ8
```