Michelle Wang and Priya Rajbhandary
Professor Jeannie Albrecht
CSCI 339
12th November, 2024

Contextual Advertising

Hadoop, an open-source framework, was designed to store and process massive datasets across a distributed network of computers. The Hadoop Distributed File System (HDFS), breaks data into chunks and distributes them as well as tasks across nodes while working alongside the MapReduce model which allows programmers to focus on data processing without getting bogged down managing parallelization or fault tolerance directly. Replicating data across the nodes, Hadoop ensures reliability and uninterrupted processing even in the case of network or hardware failures. It also provides monitoring tools to easily track job progress and node health within the network. The default data flow in Hadoop starts with the user providing input and output file paths, where HDFS splits and reads the files. The mapper method processes this data, writing serialized values grouped by key to the node's memory or disk. These map outputs are then retrieved over HTTP, merged, and passed to the reducer, which aggregates the data based on keys. Finally, the reducer outputs a text file or SequenceFile, organizing the mapped input by intermediate keys to the specified output path.

Our solution calculates the click-through rate (CTR) of ads using the MapReduce programming model. The CTR is the percentage of impressions (views) of an ad on a specific page URL that led to a click. We implement this using one map and two reduce functions. Given two input files, a log of impressions and a log of clicks, the Hadoop framework processes each line from these files as the value in the input key-value pairs for our map function (where the key is a character count, unused in our solution).

In the map function, we group impressions and clicks by their impression IDs. The map function outputs each impression ID as the key, with a list containing either just the impression or both the impression and click as the value.

In the first reduce function, we use the output of the map function to extract the ad ID and page URL (or "referrer") for each entry, setting the key as the combination of ad ID and referrer. The value is either "1" if there is both an impression and click or "0" if there is only an impression, and this value indicates whether the ad was clicked or not. One thing to note is that click entries do not contain the "referrer" field, so in these cases, we simply take the referrer from the impression entry in the same value list. Because we don't know which order the click and impressions were processed and grouped with the impression ID, our reduce function first iterates through the values and stores both the impression and click before extracting the ad ID and referrer.

In the second reduce function, we aggregate these key-value pairs by summing over the values for each key. This allows us to calculate the click-through rate by dividing the total clicks by the total impressions for each ad and page URL combination. Finally, we output the ad ID and page URL as the key, with the calculated CTR as the value.

Hadoop offers many advantages with its scalability and fault tolerance, to handle and process massive datasets across distributed nodes. Firstly. data is automatically replicated to prevent loss from hardware failures, and secondly, its use of commodity hardware, acts as a cost-effective tool to handle both structured and unstructured data. We found Hadoop's automatic parallelization particularly impressive, as it efficiently enables large-scale data processing without having the user, in this case, us, manually manage multiple files or datasets.

When working with data that exceeds the storage or processing capacity of a single node, using Hadoop in a clustered environment is far more effective than in single-node mode. A cluster is also more advantageous for multi-user settings, as resources can be allocated dynamically, and tasks prioritized while allowing for a horizontal scaling of nodes. Currently, as we are processing only two datasets in our project, single-node-mode proves to be more intuitive and user-friendly, and the cost of setting up a cluster outweighs the benefits of the cluster. Although, as our dataset volume increases, transitioning to a cluster would improve efficiency.

However, given the setup and maintenance limitations that we encountered with Hadoop, it made us think about its disadvantages as well. For low-latency tasks or in the case of network issues, Hadoop's data replication approaches to fault tolerance could result in data redundancy and high storage costs. Another limitation is ironically Hadoop's high level of abstraction while automating parallelization, which obscures a lot of details of how data is partitioned and distributed across the cluster. For users new to Hadoop, like us, this initial unintuitive interface can make it challenging to fully understand underlying processes, and to work within this key-value framework and the sequential processing of MapReduce, which could inadvertently increases the development time.

Despite these limitations, we enjoyed working with Hadoop and found the overall code very clean. As we started to understand the data flow, how data is aggregated into key-value pairs, and seeing how intermediate keys are generated and evolved through the MapReduce phases was especially rewarding. We also enjoyed the process of visualizing our data flow while drafting our design document, which helped us grasp Hadoop's strengths and potential limitations in data processing.

Overall, we think that Hadoop and MapReduce are excellent tools for efficiently processing large amounts of data. However, given the simplicity of the computation in this assignment, we found that it was actually faster to run our program on a single-node Hadoop setup rather than on a cluster. This was mainly due to the overhead involved in setting up the cluster, which outweighed the parallel processing benefits for our relatively small-scale computation. Additionally, working with the MapReduce abstraction was challenging at times. While it resembles a dictionary with key-value pairs, it operates differently in that each value is processed independently, and this abstraction was initially hard to grasp. Still, we think that Hadoop and MapReduce is a powerful concept, and we enjoyed learning how to implement it for data processing.

We found the cluster setup particularly frustrating, particularly the creation of virtual machines on Microsoft Azure, because there were many different steps and fields to enter/change, and each time we made a small mistake or missed a field, we had to delete our VM and start over. This was overall quite tedious and felt like busywork. Also, once we had our VMs set up, running the cluster was also frustrating, particularly because we had to re-setup the Hadoop file system on all 4 nodes each time we tried to run it.

Final Output snippet:

```
[21cn.com, 000aV7mTNT2JA4mGx7iMpwbLqD0Io4]	0.0
[21cn.com, 00MqnKdmS9ms90wbS8rgJQIv9EoTOw]	0.0
[21cn.com, 00Sm3jlh9cAQcbhDKEHBgQ7cJ0o4u4]	0.0
[21cn.com, 00dAqnPbhpOSM9uDfqjVIQpqV3G4Fh]	0.0
[21cn.com, 01P99XHSg9eU1nNtNxmO058LPd2qg7]	0.0
[21cn.com, 04Hwp21nGdOa5e56B1lOsG0HV0F13x]	0.0
[21cn.com, 05BwUlfhaNI6mentmqCA1Jbip7Co5l]	0.0
[21cn.com, 05D8MKmTtk5rd6I3HLUgaWKd9lsGup]	0.0
[21cn.com, 05QSQXueak5mPRH1rxLLR9jLMEDIP1]	0.0
[21cn.com, 06sJVbSkJNqXfEnxTRVrW6r9oqHpQ8]	0.0
[21cn.com, 073DUGbHBmUOpfm0WWEoORW30mOOX6]	0.0
[21cn.com, 07JIkt5htoluFBshBTrVFsrcwUTNd3]	0.5
[21cn.com, 08tOajiLVw2x115gUp5oXthD6jLOoR]	0.0
```

```
[21cn.com, 09tkspRaCqqAKhMBTv2X8aDgG44pmu]      0.6666666666666666
[21cn.com, 0AI5UbEMgfw6iiXCBhmk2v4FoiklWd]      0.0
[21cn.com, 0AV2SIN4HJPRF94l6dG7Bp5xGQ1f0V]      0.0
[21cn.com, 0CEoX9Xc5kM9Grq1pBwFoAVBuoLTAb]      0.3333333333333333
[21cn.com, 0CFO3fE8kRbkExh0HrCX5VSsiVwrFE]      0.0
[21cn.com, 0Cqkc5VEIvjIhihB4E8t8E77Lp1EOp]      0.0
[21cn.com, 0DmDiS7BLFroJ6jFvb94FclvCm0c2b]      0.3333333333333333
[21cn.com, 0F842TI9tb6TavWVxl2jLL9x0BaIRR]      0.0
[21cn.com, 0G7HTCun8oXOXEdFrPE2Kj9ePnWjNd]      0.0
[21cn.com, 0Gh5BouJWwnIWF2gSJSWnEV7xXOS1J]      0.0
[21cn.com, 0HnHLlcJWRBI58AftUWC5XoRRHvnwb]      1.0
[21cn.com, 0IfTiDJbQApAuL58fcv6S2lNFR6HqB]      0.0
[21cn.com, 0JF3P6kWLUtTWXmMUq0iwjTRX3x0Pg]      0.3333333333333333
[21cn.com, 0KmCQOVfAlqEBon4vPkg08TRASCvOa]      0.0
[21cn.com, 0Ms38uJpen6RR0IKEb3Ju8269fXsjr]      0.0
[21cn.com, 0O0IGIH1RNlgaqFxDbAIej8tKVREsx]      0.0
[21cn.com, 0O1UiCVquFNeq8TonRa5kpO2VcvJvR]      0.3333333333333333
[21cn.com, 0OWGpfM05HSi4033KPFl6O8q9nkgAF]      0.0
[21cn.com, 0P7NCHL89j4Ds39fBe9KbMd7N4kqmf]      0.0
[21cn.com, 0QkRCXxwgG2l0tLdWMueT4uwUEKJOE]      0.0
[21cn.com, 0RNPlgdkgGCDQ9B12mWbnWC4PI7fdQ]      0.0
[21cn.com, 0S8gGP63WtSfBbvlwE3kOiWb4JmUsc]      0.0
[21cn.com, 0TFoLAPksNFToFL8SJJtI2NSg1BHAT]      0.0
[21cn.com, 0TOdBtCVpLmKIsmA9m3e1NtuxBwRrL]      0.0
[21cn.com, 0TgSf17LjHcppcQIBbXres3wsTT1qO]      0.0
[21cn.com, 0U0ReNboLmQvrBxptAa43HQPH0D1b1]      0.0
[21cn.com, 0V2C5hFm35Wmqn4RFRISIMgcJrmGin]      0.0
[21cn.com, 0VlrxPeorDFEvUxFSdFIV8dgTGjd3G]      0.0
[21cn.com, 0XdupntWrgxntQlSePf4uxWT4JMeA3]      0.0
[21cn.com, 0aaCn8lmgPKiJVGFrB2J6FBGPxkapC]      0.0
[21cn.com, 0andaSOrUu6TgSe7iBrcqlGnJHQspJ]      0.0
[21cn.com, 0blL450X9Ud3roiLUUX3s4nHiXev4o]      0.0
[21cn.com, 0cEQKa6TxwsA4SV2pNqbwL3ssUSjPL]      0.0
[21cn.com, 0dEar84cA7wPFkIr8FmkffMd4jKnso]      0.0
[21cn.com, 0dnO64wA1ktLJ0gqioc2OTIrpjLDPr]      0.0
[21cn.com, 0eodn8rrh0QvlDd46tgIglpGii0K24]      0.0
[21cn.com, 0fMaj56jjsD29owGOAUJqFRL3IrKoe]      0.0
```