Wikipedia Trending Pages Analysis

Amir Zai, Rajesh Thallam, Shelly Stanley
UC Berkeley School of Information (iSchool) Berkeley, CA, USA

August 20, 2015

[Source Code on Github: Wikipedia Trending Page Analysis]

Abstract

Our project, "Wikipedia Trending Pages Analysis", is an analytics tool tracking trending pages in Wikipedia. Wikipedia is one of the top most visited websites on the Internet with millions of articles, 18 billion page views and 500 million unique visitors per months. We gathered and analyzed traffic data and developed a system to (i) display the top 10 trending pages on Wikipedia in last 24 hours (ii) display top 10 trending pages on Wikipedia in last 30 days (iii) predicting page view traffic for the trending pages in last 24 hours and (iv) search and view past trends for a page or article based on users interest. The tool is built on Apache Spark platform hosted on HDFS and Hive with the end results visualized on Kibana hosted on ElasticSearch index. The project has successfully demonstrated how to scale up a data pipeline to process, store and analyze batches and streams of large data sets of volumes in terabytes and analyze traffic for approximately 3.5 million articles.

Keywords: Wikipedia; Predictions; Trend Analysis; Apache Spark; Apache Hive; Elastic Search; Kibana; SoftLayer.

Motivation and Prior Work

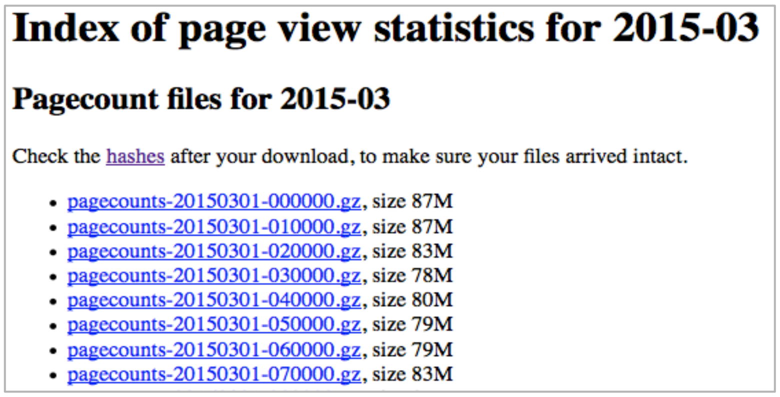
The project is an inspiration from the Wikipedia hourly page view statistics data hosted on Amazon Public Data Sets [1]. This is a rich data sett that includes 16 months of hourly page traffic statistics for over 2.5 million Wikipedia articles along with associated content, link graph and metadata. There have been multiple projects in the past utilizing the same or similar data sets to identify the trends on Wikipedia pages, for example trendingtopics.org[2] and wikirank [3] (both the websites are no more active). This dataset is recently being used for specific prediction scenarios such as predicting box office success[4], predicting flu using Wikipedia[5] data. There is a case being made that predictions using Wikipedia traffic data is more accurate than Google traffic data[6], which further emphasizes importance of this rich public data set.

Our project is inspired from the prior work mentioned, however, it is different in the following ways (i) demonstrate how the hardware and software can make a solution scalable (ii) making solution more real time (iii) showing real time predictions by embedding naïve prediction algorithms within the data pipeline. We have not observed prior implementations taking both scalability and real time predictions into account. Our work demonstrates how data processing and analytics frameworks such as Apache Spark, Apache Hive, Openstack Swift and

ElasticSearch solve the scalability problem when combined with infrastructure effectively. The paper is organized in following sections – section I covers data sources and characteristics, section II covers implementation approach and cluster architecture, section III covers data acquisition phase, section IV covers data processing phase, section V covers prediction and section VI covers trend analysis and results and the rest of the sections covers challenges faced and future improvements.

I. Data Sources and Characteristics

We used hourly Wikipedia traffic data available from Wikimedia downloads[7] which records each request of a page and collects Wiki project name, title of the page requested, size of the page requested and total page requests. This data is available since 2007 to the present date. However, for the project we only used data from January 2015 to August 2015. The volume of the data is 650GB in compressed format (gzip) for 3 million articles (approx.). The files are available at hourly frequency with each file up to ~100MB. Here is a sample of data files for March 2015 from downloads.



Index of page view statistics for 2015-03

Pagecount files for 2015-03

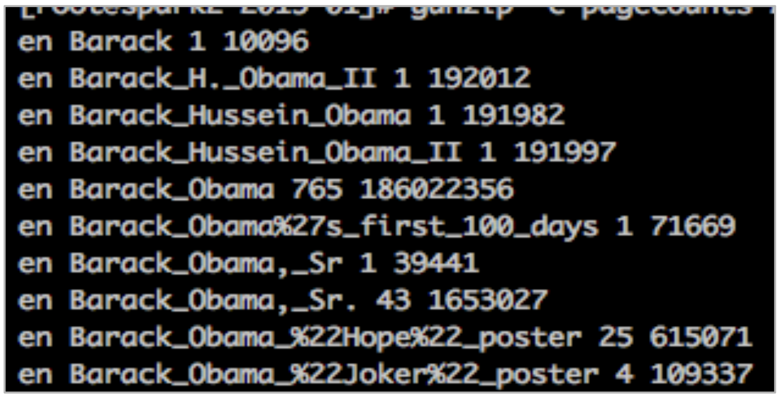
Check the hashes after your download, to make sure your files arrived intact.

- pagecounts-20150301-000000.gz, size 87M
- pagecounts-20150301-010000.gz, size 87M
- pagecounts-20150301-020000.gz, size 83M
- pagecounts-20150301-030000.gz, size 78M
- pagecounts-20150301-040000.gz, size 80M
- pagecounts-20150301-050000.gz, size 79M
- pagecounts-20150301-060000.gz, size 79M
- pagecounts-20150301-070000.gz, size 83M

Figure 1 Raw Data Files Download Snapshot

Each file name has year, month, date and hour when the statistics were collected. Each hourly traffic statistics files is in the below format

```
Project name, Page Title, Page Views in the hour, Page Size
```



```
en Barack 1 10096
en Barack_H._Obama_II 1 192012
en Barack_Hussein_Obama 1 191982
en Barack_Hussein_Obama_II 1 191997
en Barack_Obama 765 186022356
en Barack_Obama%27s_first_100_days 1 71669
en Barack_Obama,_Sr 1 39441
en Barack_Obama,_Sr. 43 1653027
en Barack_Obama_%22Hope%22_poster 25 615071
en Barack_Obama_%22Joker%22_poster 4 109337
```

Figure 2 Snapshot of Raw Data File

The same data can be represented in tabular format. For the purpose of the project, we did not use "Page Size" in the analysis.

Other data sources in the project include Wiki page lists and redirects list data to cleanup the page redirects (more in section III). This data is used only during the data-cleansing phase.

II. Architecture

Before we embarked on finalizing the implementation approach, our final goals/vision of the project were (i) ability to process and analyze historical batch data and hourly streaming data (ii) scalable platform to support the big data analysis and (iii) less time to market implementation as our project was expected to be complete within 7 weeks. The overall implementation and system architecture included following phases – (i) Infrastructure setup (ii) Data acquisition (iii) Data cleansing, processing and storage (iv) Data Analysis and (v) Presentation of results.

Figure 3 depicts the system architecture of the project

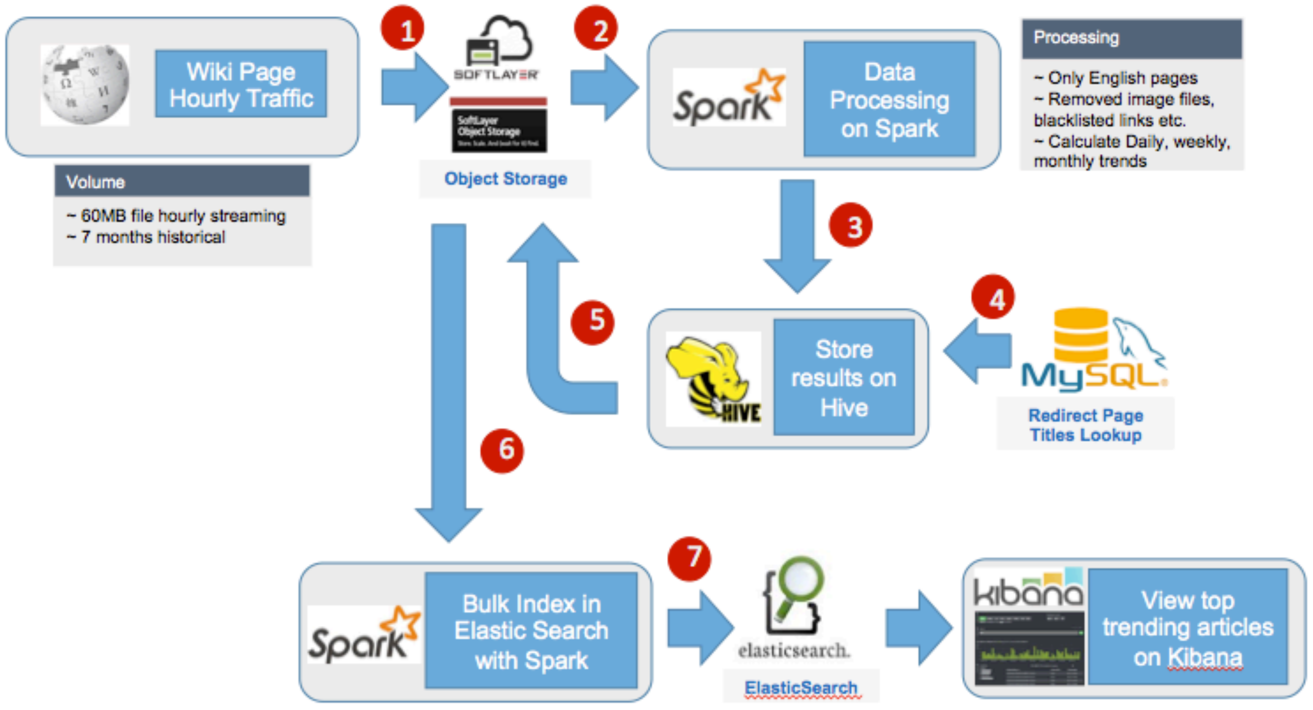


Figure 3 System Architecture

The implementation starts from (1) building a data acquisition phase to *ingest hourly Wikipedia page traffic statistics data* starting January'2015 till data. (2) This data is *acquired into SoftLayer object storage based on Openstack Swift*. (3) Page traffic data in the object storage layer is *cleansed and processed by* `Apache Spark` *using* `pyspark` to filter non-English pages, cleanse page titles and (4) *clean page redirects using* `MySQL` *dumps* from Wikimedia. (5) The resultant data is used for trend calculation at daily, weekly and monthly level for each Wikipedia page. The *resultant data from analysis phase is pushed into* `Apache Hive` data store based on HDFS. During this phase, current trends are merged with historical trends. For the top 10 trending pages in last 24 hours, a naïve *prediction algorithm runs to forecast page traffic*.

(6) The *results of the trend analysis and predictions are fed into* `ElasticSearch` *database for indexing*. (8) Finally the *results are presented to the user by integrating ElasticSearch index with* `Kibana visualization`.

To support the implementation architecture, the project has the following infrastructure setup shown in Figure 4. SoftLayer Salt CLI is used for the cluster setup and configuration.

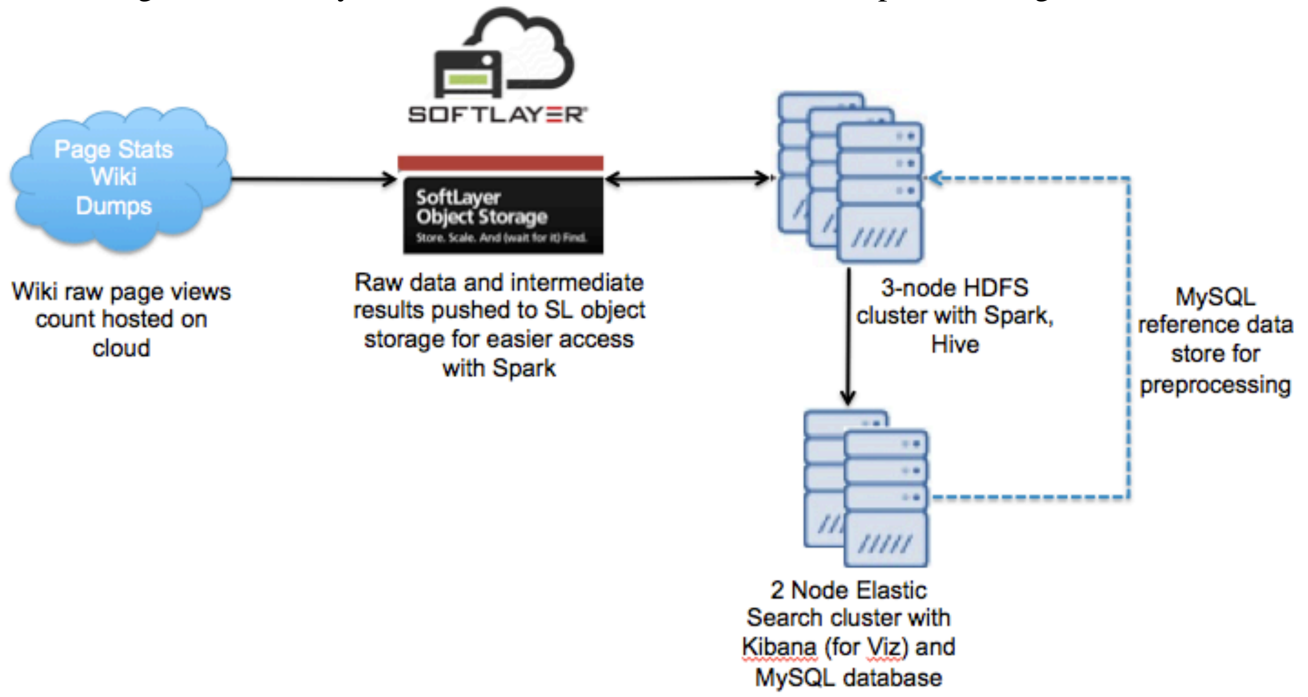

Page Stats Wiki Dumps
Wiki raw page views count hosted on cloud
SOFTLAYER
SoftLayer Object Storage
Store. Scale. And (wait for it) Find.
Raw data and intermediate results pushed to SL object storage for easier access with Spark
3-node HDFS cluster with Spark, Hive
MySQL reference data store for preprocessing
2 Node Elastic Search cluster with Kibana (for Viz) and MySQL database

Figure 4 System Infrastructure

The raw data from Wikimedia dumps is stored in SoftLayer Object Storage located in San Jose data center to maintain data locality with the data processing.

The core data processing is done on a 3-node cluster with each node having 4 cores, 8 GB of available memory and 100 Mbps network speed located on San Jose data center. HDFS is used as a shared file system to store the intermediate results and final analysis results stored on the Apache Hive warehouse. Apache Spark 1.4.1 is deployed across the entire cluster to minimize network traffic for source files. Due to write affinity with HDFS, process executes all the transformation and data cleansing scripts on the appropriate node by integrating raw data from SoftLayer object storage. Apache Spark job submissions were configured to use 4 GB of memory per executor and driver.

The final trend analysis and predictions results were stored on a 2-node ElasticSearch cluster with default 5 shards and similar node configuration as core processing engine i.e. 4 cores, 8 GB of available memory and 100 Mbps network speed located on San Jose data center. The index size estimated for ~3 million articles for 1 month is ~10GB with an average index record length of 100 bytes. To manage ElasticSearch cluster we used ElasticSearch head plugin. Kibana visualization hosted on the same nodes as ElasticSearch is the presentation layer for the project.

MySQL data store with page and redirects information is used for data cleansing. This small data store is hosted on the ElasticSearch cluster nodes

III. Data Acquisition – Ingestion and Cleansing

(a) Data Ingestion

Wikipedia page view traffic data from Wikimedia dump is available at hourly frequency with date and time available on the file name and this data is not available within the file. Data ingestion used shell script to `wget` the files from Wikimedia dumps and upload them to SoftLayer Swift object storage using `python-swiftclient`. To acquire the historical data from January'2015 to August'2015, roughly of volume 650GB compressed, we utilized the 3-node Apache Spark cluster with multiple CPU cores and 100 Mbps network speeds to efficiently complete the process within 24 hours spread across 3 days. For streaming the hourly data, the shell script downloads one file every hour or 24 files every day (one file per hours) on a single node Apache Spark cluster and later to merge with historical trends.

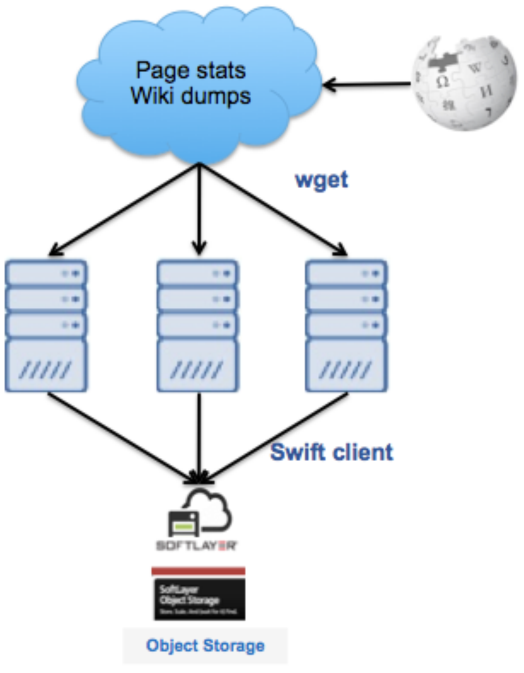


Figure 5 Data Ingestion

```
wget -q -P ${download_dir}
${download_url_file}
swift upload ${swift_container} ${file}
```

Swift object storage as the raw data acquisition layer was not the original choice of the project. We started off with HDFS based shared file system to store the data but we soon ran into the "Exceeded maximum files open issue" while processing historical data. The error continued to persist even after changing the file system limits (and rebooting the machines). Based on the raw data volume metrics we realized HDFS must have at least 900 GB of disk space. Considering the data volumes, we switched from HDFS based data acquisition layer to Swift object storage based layer. Switching to swift object storage layer, the maximum files open issue ceased to exist. This setup further benefitted the processing layer with Apache Spark where each layer downloaded required data from Object Storage to work with.

(b) Data Cleansing

After performing the cursory data analysis we came to a conclusion to keep only English related pages, i.e. projects starting `'en'`, from the page view statistics to avoid the any noise (encode/decode) in non-English pages. Following cleansing activities are performed on the page titles – (i) exclude page titles starting with lower case (ii) exclude page titles related to image, (iii) limit pages that belong to Wikipedia main namespace (namespace 0), (iv) cleaning redirected pages within the same Wikipedia page (i.e. Facebook and Facebook#timeline are same pages) and (v) remove blacklisted pages such as 404, Wiki main page, search pages etc.

After the source data is uploaded to Swift Object Storage, 3-node Apache Spark cluster reads each file using Hadoop Openstack Swift adapter and performs afore said cleansing activities. The cleansed or standardized page title from each file is combined with date, time and page view count from the file and fed to the data processing stage.

IV. Data Processing

After the data is cleaned and standardized, we did basic processing at this stage to calculate daily, weekly, and monthly trends. The results were generated in this format.

```
Page Title | List of Dates | Page Views | Daily Trends | Weekly Trends | Monthly Trends

Barack_Obama|20150102,20150122,20150123,20150126,20150130|1,1,3,2,1|0.0,0.0,2.0,-
0.333333333333,-0.5|0.0|0.0|

George_Langton_Hodgkinson|20150115,20150119,2
0150121|1,1,1|0.0,0.0,0.0|0.0|0.0|
```

The processing is performed on Apache Spark cluster and results are written to HDFS shared file system. The resultant files are concatenated and stored in Apache Hive data store for further processing. Before using the resultant trends for analysis, page titles must be further cleansed to correct for redirects. We merged page titles and page redirects MySQL export from Wikimedia dumps to create a reference table to match page title from Wikipedia page traffic data and return the true title. For example, all the following pages under the redirect title map to the true title

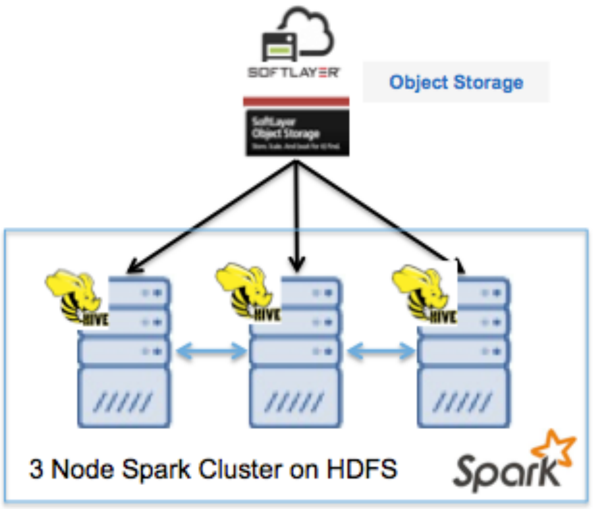
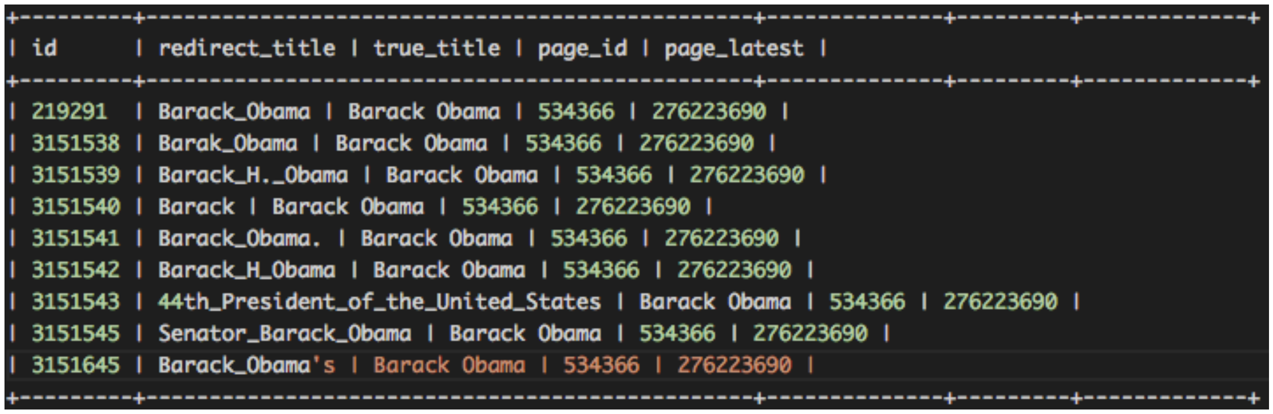


Figure 6 Data Processing on Spark and Hive



```
+---------+--------------------------------------------+-------------+-----------+-------------+
| id      | redirect_title | true_title | page_id | page_latest |
+---------+--------------------------------------------+-------------+-----------+-------------+
| 219291  | Barack_Obama | Barack Obama | 534366 | 276223690 |
| 3151538 | Barak_Obama | Barack Obama | 534366 | 276223690 |
| 3151539 | Barack_H._Obama | Barack Obama | 534366 | 276223690 |
| 3151540 | Barack | Barack Obama | 534366 | 276223690 |
| 3151541 | Barack_Obama. | Barack Obama | 534366 | 276223690 |
| 3151542 | Barack_H_Obama | Barack Obama | 534366 | 276223690 |
| 3151543 | 44th_President_of_the_United_States | Barack Obama | 534366 | 276223690 |
| 3151545 | Senator_Barack_Obama | Barack Obama | 534366 | 276223690 |
| 3151645 | Barack_Obama's | Barack Obama | 534366 | 276223690 |
+---------+--------------------------------------------+-------------+-----------+-------------+
```

Figure 7 Page redirects lookup from MySQL

This translation was necessary to display the trends accurately for the redirect pages as the source traffic data from Wikimedia considers these redirects as unique pages even though they redirect to the same page.

The Spark jobs for the historical batch and hourly streaming batch were submitted as below

```
$SPARK_HOME/bin/spark-submit --executor-memory 4G --driver-memory 4G --driver-cores 2 --executor-cores 2 --num-executors 4 --jars $HADOOP_HOME/share/hadoop/tools/lib/hadoop-openstack-2.6.0.jar --master spark://spark01:7077 wiki_pageview_trends.py
```

The spark jobs are supplemented with an additional jar file customized to read source data from SoftLayer object storage. Figure 8 and Figure 9 depicts the DAG (Direct Acyclic Graph) [8] visualization from Spark cluster showing the chain of RDD dependencies while processing single hourly file processing and 24 hourly files for daily load processing.

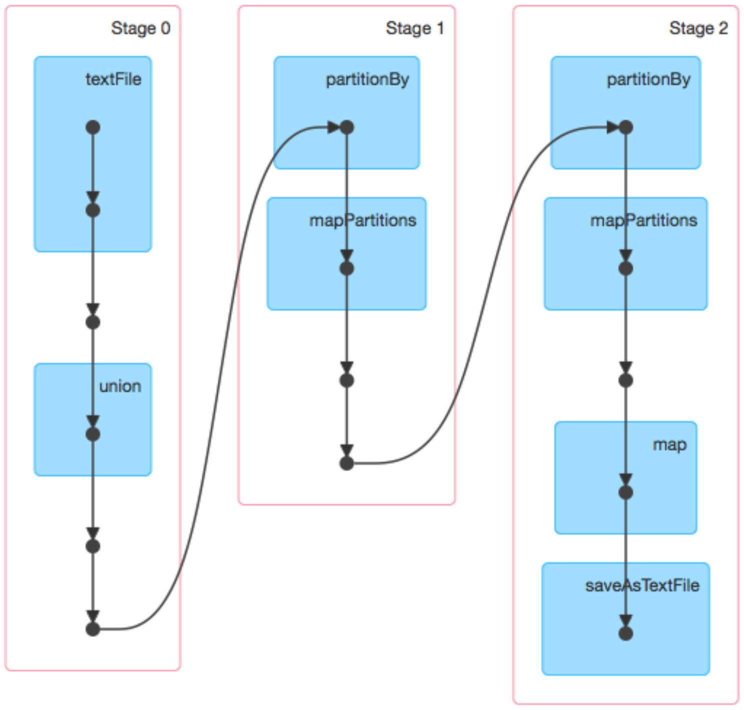


Figure 8 DAG Visualization for Hourly File

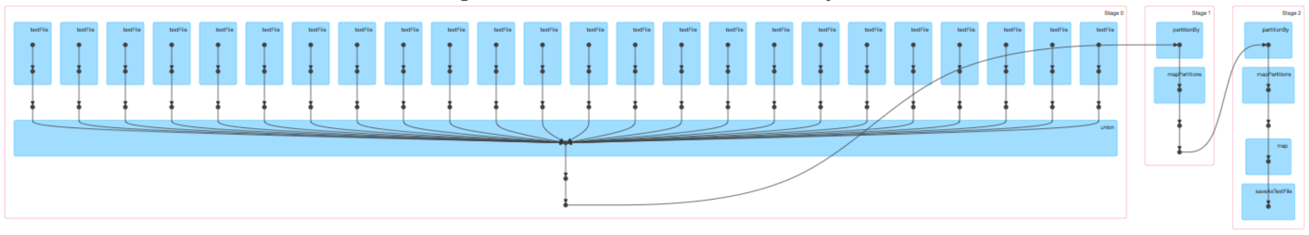


Figure 9 DAG Visualization for 24 hourly files for daily load processing

Figure 10 shows the event timeline associated with daily load processing which helps to identify any bottlenecks within an application and also how the executors dynamically allocated by the master based on the workload (1 file vs. 24 files).

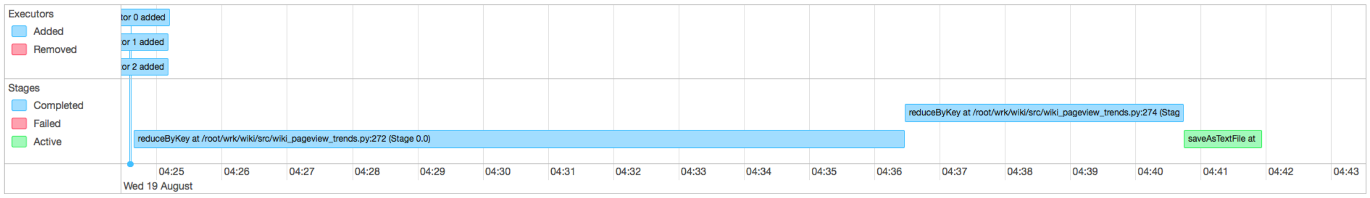
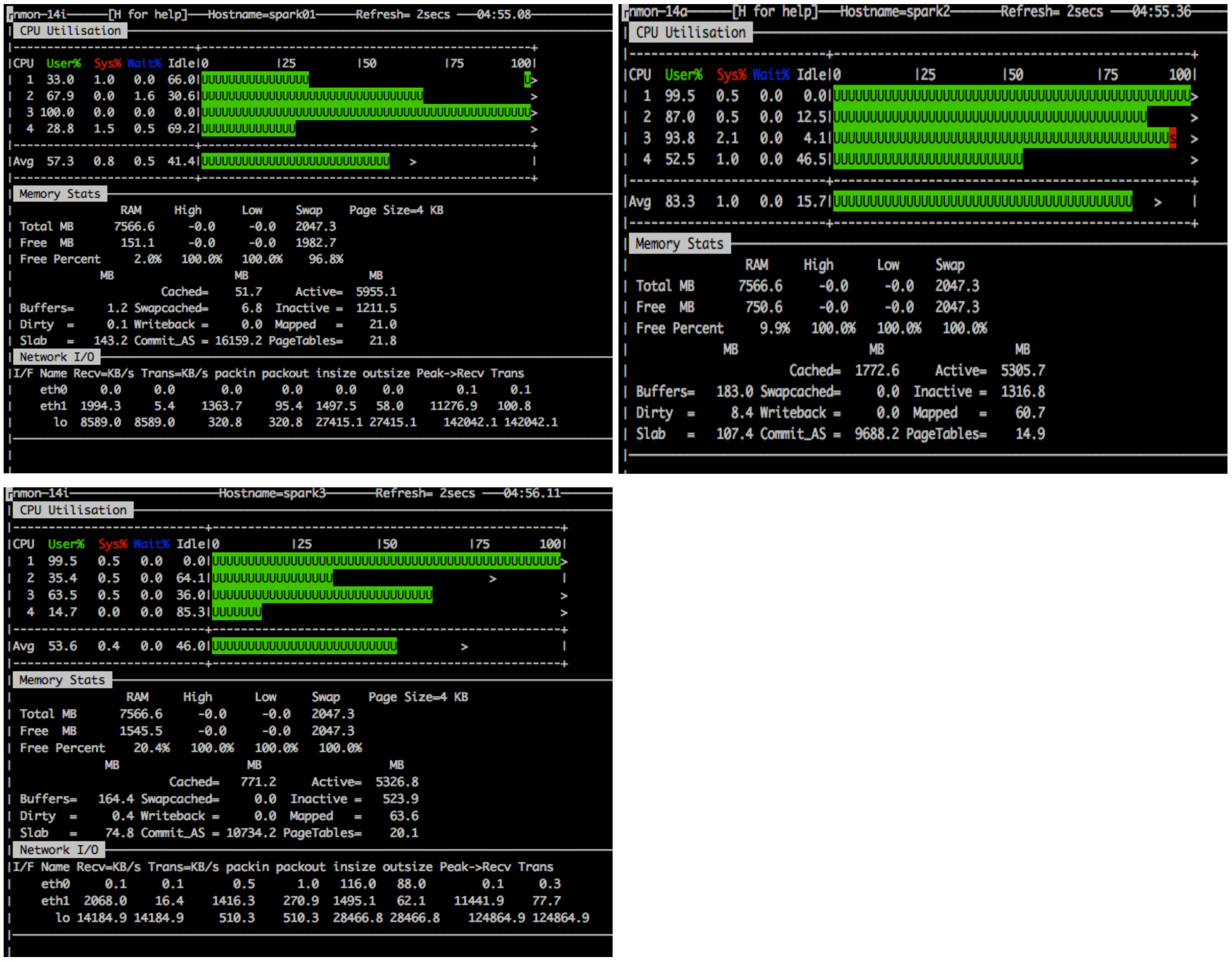


Figure 10 Event Timeline of a Spark application

To monitor the network, CPU and memory we installed nmon [9]on all the three nodes of spark cluster which allowed us to identify the I/Os, CPU utilization and configure spark context configuration accordingly. Figure 11 shows snapshot captured for a daily files processing job.



```
┌nmon─14i─────[H for help]───Hostname=spark01──────Refresh= 2secs ───04:55.08──
│ CPU Utilisation ─────────────────────────────────────────────────────────────
│---------------------------+--------------------------------------------------+
│CPU  User%  Sys% Wait% Idle│0          │25          │50          │75       100│
│  1  33.0   1.0   0.0  66.0│UUUUUUUUUUUUUUUU                                 U>
│  2  67.9   0.0   1.6  30.6│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU                >
│  3 100.0   0.0   0.0   0.0│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU>
│  4  28.8   1.5   0.5  69.2│UUUUUUUUUUUUUU                                    >
│---------------------------+--------------------------------------------------+
│Avg  57.3   0.8   0.5  41.4│UUUUUUUUUUUUUUUUUUUUUUUUUUUUU   >                 │
│---------------------------+--------------------------------------------------+
│ Memory Stats ────────────────────────────────────────────────────────────────
│              RAM     High      Low     Swap    Page Size=4 KB
│ Total MB     7566.6    -0.0      -0.0   2047.3
│ Free  MB      151.1    -0.0      -0.0   1982.7
│ Free Percent    2.0%  100.0%    100.0%    96.8%
│             MB                  MB                  MB
│                      Cached=    51.7     Active=  5955.1
│ Buffers=      1.2 Swapcached=    6.8  Inactive =  1211.5
│ Dirty  =      0.1 Writeback =    0.0  Mapped   =    21.0
│ Slab   =    143.2 Commit_AS = 16159.2 PageTables=   21.8
│ Network I/O ─────────────────────────────────────────────────────────────────
│I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Recv Trans
│    eth0      0.0       0.0      0.0      0.0     0.0     0.0        0.1     0.1
│    eth1   1994.3       5.4   1363.7     95.4  1497.5    58.0    11276.9   100.8
│      lo   8589.0    8589.0    320.8    320.8 27415.1 27415.1   142042.1 142042.1
│──────────────────────────────────────────────────────────────────────────────
│
```

```
┌nmon─14a─────[H for help]───Hostname=spark2──────Refresh= 2secs ───04:55.36──
│ CPU Utilisation ─────────────────────────────────────────────────────────────
│---------------------------+--------------------------------------------------+
│CPU  User%  Sys% Wait% Idle│0          │25          │50          │75       100│
│  1  99.5   0.5   0.0   0.0│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU>
│  2  87.0   0.5   0.0  12.5│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU      >
│  3  93.8   2.1   0.0   4.1│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUS  >
│  4  52.5   1.0   0.0  46.5│UUUUUUUUUUUUUUUUUUUUUUUUUU                        >
│---------------------------+--------------------------------------------------+
│Avg  83.3   1.0   0.0  15.7│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU   >    │
│---------------------------+--------------------------------------------------+
│ Memory Stats ────────────────────────────────────────────────────────────────
│              RAM     High      Low     Swap
│ Total MB     7566.6    -0.0      -0.0   2047.3
│ Free  MB      750.6    -0.0      -0.0   2047.3
│ Free Percent    9.9%  100.0%    100.0%   100.0%
│             MB                  MB                  MB
│                      Cached=  1772.6     Active=  5305.7
│ Buffers=    183.0 Swapcached=    0.0  Inactive =  1316.8
│ Dirty  =      8.4 Writeback =    0.0  Mapped   =    60.7
│ Slab   =    107.4 Commit_AS =  9688.2 PageTables=   14.9
│──────────────────────────────────────────────────────────────────────────────
```

```
┌nmon─14i─────────────────────Hostname=spark3──────Refresh= 2secs ───04:56.11──
│ CPU Utilisation ─────────────────────────────────────────────────────────────
│---------------------------+--------------------------------------------------+
│CPU  User%  Sys% Wait% Idle│0          │25          │50          │75       100│
│  1  99.5   0.5   0.0   0.0│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU>
│  2  35.4   0.5   0.0  64.1│UUUUUUUUUUUUUUUUU                      >          │
│  3  63.5   0.5   0.0  36.0│UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU                   >
│  4  14.7   0.0   0.0  85.3│UUUUUUU                                           >
│---------------------------+--------------------------------------------------+
│Avg  53.6   0.4   0.0  46.0│UUUUUUUUUUUUUUUUUUUUUUUUUU         >              │
│---------------------------+--------------------------------------------------+
│ Memory Stats ────────────────────────────────────────────────────────────────
│              RAM     High      Low     Swap    Page Size=4 KB
│ Total MB     7566.6    -0.0      -0.0   2047.3
│ Free  MB     1545.5    -0.0      -0.0   2047.3
│ Free Percent   20.4%  100.0%    100.0%   100.0%
│             MB                  MB                  MB
│                      Cached=   771.2     Active=  5326.8
│ Buffers=    164.4 Swapcached=    0.0  Inactive =   523.9
│ Dirty  =      0.4 Writeback =    0.0  Mapped   =    63.6
│ Slab   =     74.8 Commit_AS = 10734.2 PageTables=   20.1
│ Network I/O ─────────────────────────────────────────────────────────────────
│I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Recv Trans
│    eth0      0.1       0.1      0.5      1.0   116.0    88.0        0.1     0.3
│    eth1   2068.0      16.4   1416.3    270.9  1495.1    62.1    11441.9    77.7
│      lo  14184.9   14184.9    510.3    510.3 28466.8 28466.8   124864.9 124864.9
│──────────────────────────────────────────────────────────────────────────────
│
```

Figure 11 Network, CPU and Memory Usage of Spark Cluster

To merge the current trends with the historical trends we used Hive QL (Query Language) joins and recalculated trends. We preferred to use Apache Hive for this processing instead of Spark SQL data frames considering the huge volume of results are stored in Hive and using spark would have caused I/O issues. In fact we saw performance being slow on Spark SQL data frame compared to Hive QL on a month worth of files. Spark SQL data frames [10]need to be further explored to confirm the issue. The final results were pushed to SoftLayer object storage for future retrieval, if needed.

V. Predicting Trends

Wikipedia pages with the daily, weekly and monthly trends are exported to a flat file from Hive data store. This data is transformed to a format appropriate for prediction algorithms using machine learning. Every row in the resulting dataset represents the normalized backward (daily and weekly) and forward (only daily) trends for a page. The date and page column in the file is removed to create a model effectively agnostic to date and page-specific characteristics.

Including these features results in a model overly specific and overfits to the data. In order to leverage such information less granular features need to be extracted. For instance whether a date is a holiday and page category could provide valuable signal and yield more accurate predictions.

A few instances of the dataset used for training the machine-learning model:

Past Day Trend	**Past Week Trend**	**Next Day Trend**
0.1	0.05	0.12
-0.02	0.19	0.09
...	...	...

To calculate the features mentioned previously, following formulae are applied:

```
Past Day Trend  = Yesterday’s views for the page / Today’s views - 1
Past Week Trend = Views from 7 days ago / Today’s views - 1
Next Day Trend  = Next day’s views / Today’s views - 1
```

We trained a Random Forest regression model[11] using Spark MLLib’s ensemble library. The R^2, percentage of variance in dependent variable (predicted trend) explained by independent variables (feature) is 0.67. The following default parameters were used for training the model:

- numTrees=3
- featureSubsetStrategy="auto"
- iimpurity='variance'
- maxDepth=4
- maxBins=32

We can now use this model to predict the next day traffic. For instance we can run all the data for January 14, 2015 and sort the data by predicted traffic to effectively find the pages that will be trending the next day based on historical trends. The following pages are predicted to be trending based on relative traffic growth:

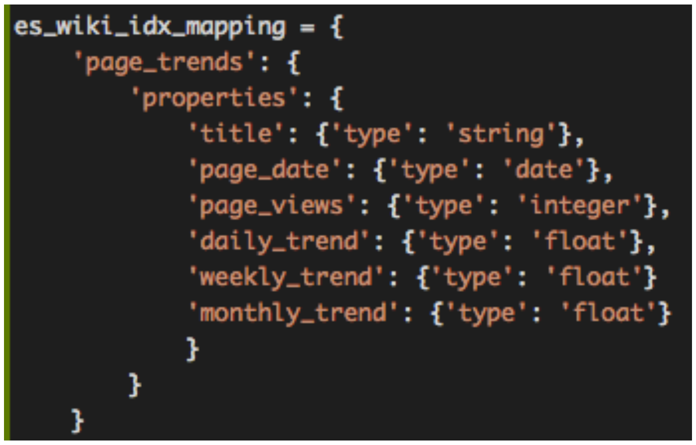
Page	**Predicted Next Day Trend**
American_Sniper_(film)	4.29
Transparent_(TV_series)	3.85
Christiano_Ronaldo	2.92
Edward_Norton	1.84

Genghis_Khan	1.72

VI. Visualizing Trends

We used Kibana[12] on ElasticSearch[13] as the visualization platform because this combo of software is an easier choice to setup with minimum front-end development. ElasticSearch is a search server based on Lucene providing a distributed, multi-tenant capable full text search engine with a RESTful web interface and JSON documents. We predicted an index size of 80GB for 8 months of data. To host this volume of index we have set up a 2-node ElasticSearch cluster monitored by ElasticSearch head plugin.

We have finalized three index types under "wiki" ElasticSearch database to (i) search for trending pages in last 24 hours (ii) search for trending pages in last 30 days and (iii) view predictions for top-N trending pages. Figure 12 shows snapshot if ElasticSearch index structure.



```
es_wiki_idx_mapping = {
    'page_trends': {
        'properties': {
            'title': {'type': 'string'},
            'page_date': {'type': 'date'},
            'page_views': {'type': 'integer'},
            'daily_trend': {'type': 'float'},
            'weekly_trend': {'type': 'float'}
            'monthly_trend': {'type': 'float'}
        }
    }
}
```

Figure 12 ElasticSearch index format

To build the ElasticSearch index, we used Apache Spark combined with ElasticSearch bulk index helper in python client. We were able to build index in ~2 hours for one month of trends. Without using Apache Spark the index build takes ~ 3.5 hours for one month of trends.

Kibana is more than a data visualization plugin for ElasticSearch. It integrates seamlessly with ElasticSearch and effectively produces visualizations when time series data is involved. We have developed four visualizations to achieve our final goal and stitched all of them on to a Kibana dashboard.

(i) Search for trend analysis for any Wikipedia page starting from 2015
(ii) View top-N (top-10) trending pages in last 30 days
(iii) View top-N (top-10) trending pages in last 24 hours
(iv) View page view count predictions for trending pages in last 24 hours

Figure 14 shows the final Kibana dashboard setup for the project.

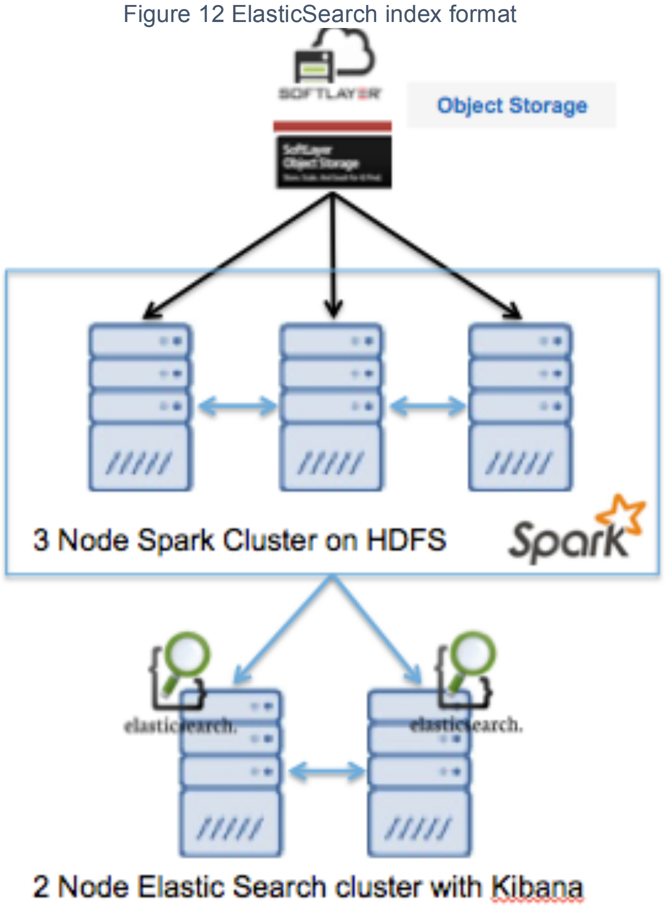
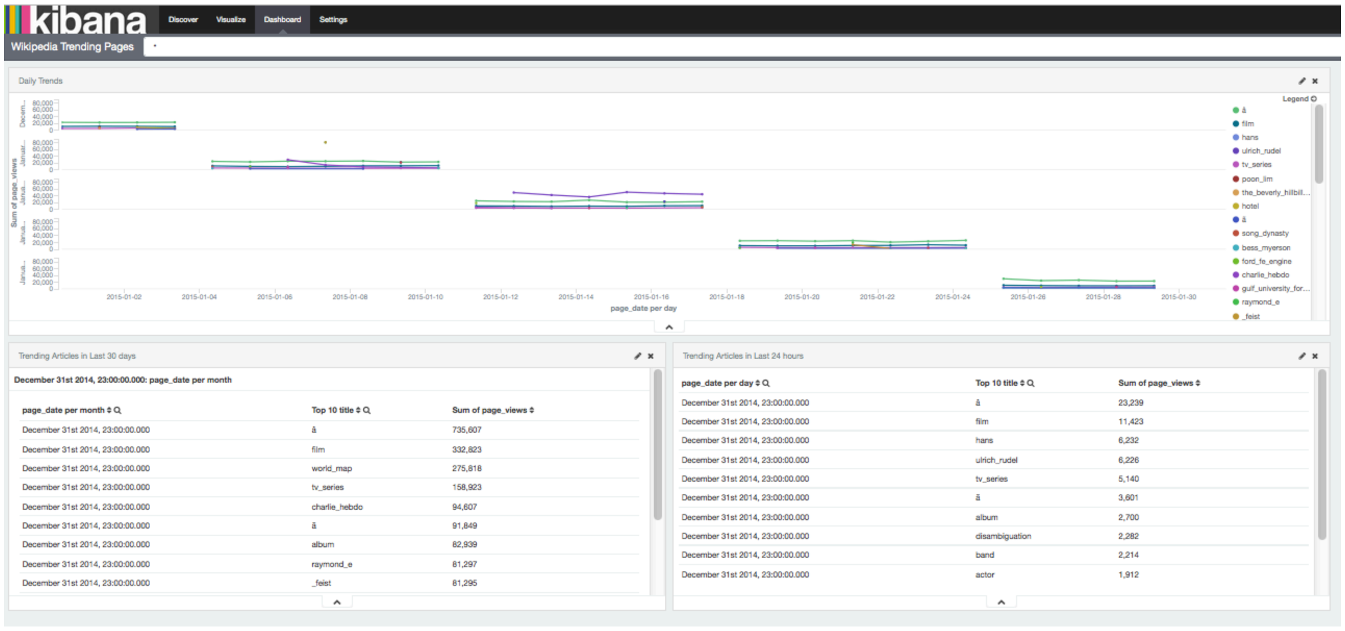


Figure 13 ElasticSearch and Kibana Setup for Results



Trending Articles in Last 30 days

December 31st 2014, 23:00:00.000: page_date per month

page_date per month	Top 10 title	Sum of page_views
December 31st 2014, 23:00:00.000	â	735,607
December 31st 2014, 23:00:00.000	film	332,823
December 31st 2014, 23:00:00.000	world_map	275,818
December 31st 2014, 23:00:00.000	tv_series	158,923
December 31st 2014, 23:00:00.000	charlie_hebdo	94,607
December 31st 2014, 23:00:00.000	ã	91,849
December 31st 2014, 23:00:00.000	album	82,939
December 31st 2014, 23:00:00.000	raymond_e	81,297
December 31st 2014, 23:00:00.000	_feist	81,295

Trending Articles in Last 24 hours

page_date per day	Top 10 title	Sum of page_views
December 31st 2014, 23:00:00.000	â	23,239
December 31st 2014, 23:00:00.000	film	11,423
December 31st 2014, 23:00:00.000	hans	6,232
December 31st 2014, 23:00:00.000	ulrich_rudel	6,226
December 31st 2014, 23:00:00.000	tv_series	5,140
December 31st 2014, 23:00:00.000	ã	3,601
December 31st 2014, 23:00:00.000	album	2,700
December 31st 2014, 23:00:00.000	disambiguation	2,282
December 31st 2014, 23:00:00.000	band	2,214
December 31st 2014, 23:00:00.000	actor	1,912

Figure 14 Kibana Dashboard for Wikipedia Trending Pages

VII. Challenges

1. Too much noise in the source file

As someone said, Data Scientists job is 80% cleansing data and 20% is analyzing and exploring the data. True to the statement our project spent considerable time in standardizing and filtering the data for accurate predictions and analysis. It started from the date and timestamp on the traffic statistics file that was available on the file name instead of part of file content. This meant file name has to be parsed and then merged with the contents. We noticed ~20% of data was related blacklisted pages and non-standard Wikipedia pages. These pages were filtered along with same page redirects and not-same page redirects which involved additional data sources.

2. Error max files open

We started historical batch files processing that involved 24 hourly files everyday for one month i.e. ~720 files. This caused Spark job to fail due to HDFS issue related to infamous "Too Many Open Files". We configured operating system limits to high values but that did not stop the error. (Yes, we did reboot the system and ensured settings are effective). We soon knew that HDFS may not be the correct option to store the raw files as we were planning to collect ~650GB compressed data format apart from intermediate results. For a scalable implementation, we went to store raw files from Wikimedia dumps into SoftLayer Object Storage using swift. This meant cheaper cost per GB and availability of files any time. This decision also benefitted Spark jobs to run on worker nodes in parallel independently.

3. Spark, Hadoop and Swift integration

Spark, Hadoop and Swift are already integrated via Hadoop Openstack library included in the distribution. But this integration requires Keystone based authentication, which is not the mechanism for SoftLayer Object Storage. SoftLayer uses URL authentication with user name/account name and API key. After searching on Google we found that a patch must be applied on the Hadoop 2.6.0 distribution to integrate SoftLayer Object Storage and Hadoop[14]. After applying the patch, this integration was transparent and seamless.

4. Not so friendly Kibana

Kibana is a seamless plugin to ElasticSearch and more than a data visualization tool. However, Kibana had minimal options to visualize the Wikipedia data set we are working with. To emphasize, one of the issue was not able to customize x-axis and y-axis. We also realized late that ElasticSearch does not detect date type automatically and the index structure must have date data type defined.

5. Spark Union operator

sc.union operator in spark is a lazy transformation. We had to use this transformation to combine RDDs for hourly files after parsing the file name contents. Initially we used sc.union as part of the for loop iterator over the available files. This caused operation to be sequential and time consuming. We upgrade from Spark 1.3.1 to Spark 1.4.1 and after debugging through Spark Event timelines and DAG visualizations we found the issue

6. MySQL reference data store

We used MySQL export dumps for finding redirect title and true title. However, the huge volume of data in MySQL joining with Hive data sets was time consuming. The approach was to export this data to Apache Hive and store reference data along with the intermediate results. We observed that running MySQL imports was time taking. So instead of importing the MySQL dumps into the MySQL database, we build a PERL utility to read the MySQL files and convert it as text file that can be directly imported into Hive.

7. Spotify's Snakebite client for HDFS

Since our project uses HDFS as shared file system, we were looking for a python client that can integrate and perform well with HDFS. The native HDFS client invokes a JVM process, which is often slower even for basic commands. We resorted to Spotify's Snakebite [15]client that provides a pure python HDFS client using protocol buffers (RPC) to communicate with the name node. This meant Snakebite client communication with HDFS is much faster than native APIs available such as libhdfs, Httpfs etc. Snakebite has limited features compared to the Hadoop HDFS communication but it was sufficient for the project to work with.

VIII. Future Improvements

There is no project without any future improvements and so is ours. There is a potential continuation with this project

1. Additional sources for better prediction

There is so much wiki data out there and it would be valuable to gather more and different types to enhance the prediction engine. Ideas include page links, Wikipedia edits data and correlation amongst pages over the time.

2. Spark SQL data frame instead of Hive

Spark SQL data frame in Spark 1.4.1 would have been preferable because Hive's performance is less than ideal in certain cases such as aggregation operations. Though we have seen that Spark performs less than ideal when compared to Hive on inner joins with basic configuration.

3. Trend analysis for additional languages

For simplicity we didn't get dragged down in the weeds and we decided to begin our analysis only using English language pages. Obviously, there are plenty of good reasons to support other languages as well. It would be further valuable if we could see the trends geographically.

4. Dynamic cluster balancing

Despite having a scalable configuration we could not make the scalability aspect elastic as and when needed. Most of the time after processing hourly loads our servers were idle. If we could have used dynamic or elastic cluster balancing that would have helped at just about every stage of this project and reduce overall cost spent on the infrastructure.

5. Better user interface

Kibana was a handy tool, but would have preferred more features and a better user interface.

6. More features for more accurate prediction model

We have used basic features such as daily, weekly and monthly trends for predicting the future page view counts on top trending pages. We could have used alternate sources such as holiday calendar, page links for further analysis. Using time series analysis models such as ARIMA we could have found seasonal patterns on the page access such as on tax day or rainy day.

IX. Conclusion

We have successfully demonstrated how the project supports increasingly large systems with Spark, HDFS and SoftLayer Object Storage. The system is fully scalable and demonstrated the concepts learnt in the class 'Scaling Up! Really Big Data'

X. References

[1] Amazon Public Data Sets – Wikipedia Traffic Statistics

[2] Trending Topics using Hive and EC2

[3] Wikirank: Find what's trending on Wikipedia

[4] Predicting box office success

[5] Predicting flu outbreak using Wikipedia search data

[6] Wikipedia search data is better for predictions than Google search data

[7] Wikimedia Downloads

[8] Spark Event Timeline and DAG visualization

[9] nmon – Nigels' monitor

[10] Spark SQL and Dataframe

[11] Random Forest regression model using Spark MLlib

[12] Kibana Data Visualization tool for ElasticSearch

[13] ElasticSearch server

[14] Integrating Hadoop with SoftLayer Object Storage

[15] Spotify's Snakebite client for HDFS

Source Code on Github: Wikipedia Trending Page Analysis