Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings

Volume 17 *Boston, USA* Article 15

2017

The Billion Object Platform (BOP): a system to lower barriers to support big, streaming, spatio-temporal data sources

Devika Kakkar
Center for Geographic Analysis, Harvard University

Ben Lewis
Center for Geographic Analysis, Harvard University

David Smiley
Self Employed, Freelance Consultant

Ariel Nunez
Terranodo LLC

Recommended Citation
Kakkar, Devika; Lewis, Ben; Smiley, David; and Nunez, Ariel (2017) "The Billion Object Platform (BOP): a system to lower barriers to support big, streaming, spatio-temporal data sources," *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*: Vol. 17 , Article 15.
DOI: https://doi.org/10.7275/R5ST7N0G
Available at: https://scholarworks.umass.edu/foss4g/vol17/iss1/15

The Billion Object Platform (BOP): a system to lower barriers to support big, streaming, spatio-temporal data sources

Optional Cover Page Acknowledgements
We thank the Sloan Foundation and the Boston Area Research Initiative (BARI) for funding and Harvard Dataverse and for their collaboration. We also want to give thanks to lead developer David Smiley for his innovative contribution to the project particularly in the areas of core Lucene and Solr enhancements for spatial and temporal data handling, development of the web-service, and the design of geo-enrichment. We would like to express our gratitude to Ariel Nunez and his team at Terranodo for their innovative work on the BOP web client. We would also like to thank for the free support from Kontena that was invaluable.

The Billion Object Platform: A System to Lower Barriers to Big, Streaming, Spatio-Temporal Data Sources

Devika Kakkar[a,*], Ben Lewis[a], David Smiley[b], Ariel Nunez[c]

[a] *Center for Geographic Analysis, Harvard University 1737 Cambridge Street, Cambridge, MA, USA, 02138*

[b] *Self Employed, Freelance Consultant Cambridge, MA, USA*

[c] *Terranodo LLC San Diego, CA, USA*

Abstract: With funding from the Sloan Foundation, Boston Area Research Initiative (BARI), and Harvard Dataverse, the Harvard Center for Geographic Analysis (CGA) has developed a big spatio-temporal data visualization platform called the Billion Object Platform or "BOP". The goal of the project is to lower barriers for scholars who wish to access large, streaming, spatio-temporal datasets. Since once archived, streaming data gets big fast, and since most GIS systems don't support interactive visualization of millions of objects, a new platform is needed. Our instance of the BOP is loaded with the latest billion geo-tweets and is fed a real-time stream of about 1 million tweets per day. The CGA has been harvesting and archiving geo-tweets since 2012. Tweets flowing into the BOP are enriched with sentiment and census information to support further analysis. Incoming and intermediate data is streamed/stored in Apache Kafka. The core of the BOP is Apache Solr, which supports fast search. Some significant Solr enhancements were developed (and contributed back) – notably 2D "heatmap faceting" to support spatial visualization. The BOP fronts Solr with a RESTful web service, which provides a friendly, and secure API that is accessed from a browser-based client. The client dynamically displays temporal and spatial distributions of results for result sets containing hundreds of millions of features. The system is open source and runs on commodity hardware. The geo-tweet instance is hosted on Massachusetts Open Cloud (MOC), an OpenStack environment (OpenStack 2017). All components are deployed in Docker and orchestrated by Kontena.

[*] Corresponding author
Email address: `kakkar@fas.harvard.edu` (Devika Kakkar)

1. Introduction

1.1. Background

The Billion Object Platform (BOP) is a prototype platform designed to lower barriers for researchers who need to access big streaming spatio-temporal datasets. Most mapping systems today are extremely constrained when it comes to representing millions of features on a map unless the visualization has been previously generated (Lewis 2016a). The system provides an approach for managing real-time streaming flows of big data, and presents a prototype for storing, indexing, visualizing, and querying geo-tweets which are tweets containing a GPS coordinate from the originating device.

The system exposes the latest billion georeferenced tweets geo-tweets for exploration by the basic dimensions of time, space, keyword, and the system supports interactive visualization of distributions of tweets in both time and space. Approximately, 1-2% of tweets are geo-tweets and Centre for Geographic Analysis (CGA) at Harvard University has been harvesting them since 2012, creating an archive of about 8 billion objects to date. The BOP further enhances 2D faceting (heatmap) capabilities in Lucene and Solr that were originally developed by the CGA in a previous project to build a map services search platform called HHypermap Registry which was funded by the National Endowment for the Humanities (Lewis 2016b).

1.2. Functional Requirements

The goal of the BOP is to make it easier for a scholar to explore a big dataset and create the right subset for their research purposes. The functional requirements of the system were designed to push the envelope of what is possible in geo-visualization:

- Provide access to the most recent billion geo-tweets
- Support real-time search
- Sub-second queries including heatmaps and temporal histograms
- Implementable on the cheap using commodity servers

In addition to attempting to address a general big data visualization problem, the BOP is configured to operate within the context of a particular data archive, Harvard Dataverse, exposing an API that can be plugged into Dataverse to enable subsets of the BOP to be extracted, then pushed into Dataverse for further analysis. This instance of the BOP is loaded with tweets but a similar dataset could, with some tweaking, be substituted.

1.3. System Architecture

The high-level architecture of the BOP is shown in Figure 1 (Smiley 2016). The primary infrastructure components are Apache Kafka and Apache Solr. Tweets are harvested and then enriched with metadata in the form of sentiment as well as spatial joins with census and other canonical boundary codes. The geo-tweets are archived using a long-term Kafka topic. The BOP itself, which consists of a Solr index based copy of the data, represents the latest billion while the index goes back further in time. The BOP-core exposes a search and extraction web service API that the client consumes. All components are deployed to a Docker based infrastructure managed by Kontena. The term BOP-core is used to refer to both the Solr index and the web service that exposes it.

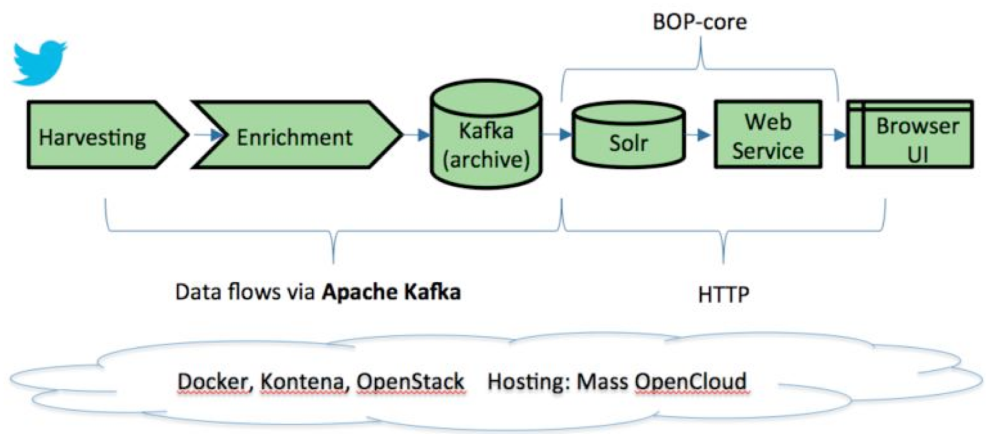

BOP-core
Harvesting
Enrichment
Kafka (archive)
Solr
Web Service
Browser UI
Data flows via **Apache Kafka**
HTTP
Docker, Kontena, OpenStack Hosting: Mass OpenCloud

Figure 1: Logical high-level architecture of BOP.

1.4. Preliminary Performance Results

The performance of the system is difficult to characterize, as there are many variables at play including: the document size of the largest Solr shard being queried, the number of docs matching a query in that shard, whether the same query has been run before (thus cached), the nature of the filters (keyword, space, time, others), and the optional components of a query: requesting top docs, requesting heatmaps (at various resolutions), requesting temporal histograms (at various resolutions), and whether or not the leading shard (for the current time period) is queried.

The table of times below has two columns; both were for separate 12-month periods (across 12 Solr shards). More data exists in the repository for 2015 than 2016. Times are in milliseconds.

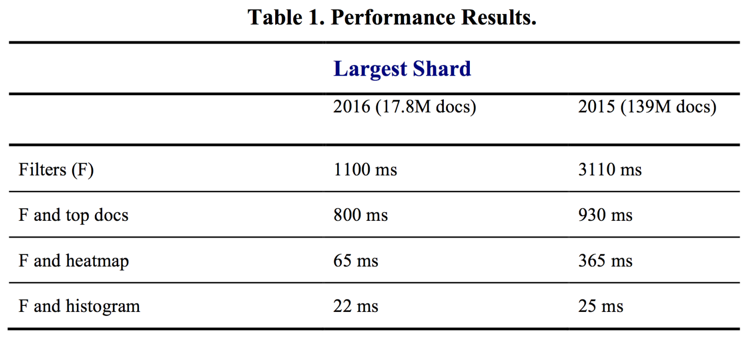


Table 1. Performance Results.

	Largest Shard	
	2016 (17.8M docs)	2015 (139M docs)
Filters (F)	1100 ms	3110 ms
F and top docs	800 ms	930 ms
F and heatmap	65 ms	365 ms
F and histogram	22 ms	25 ms

Table 1: Performance Results.

The first row is the cold (non-cached) time it took to return a simple document count for the keyword query usa along with the temporal period for that year, and a spatial filter covering the USA, northeast region. The temporal and spatial aspects were cached but the keyword was chosen anew. Performance can vary based on a number of factors. If the query has been issued before, performance is much faster. The second row is the cold (non-cached) time for returning the 25 top docs (sorted by time). Again performance can vary. If the query has been previously issued, this time is drastically reduced. The third row is the time to compute a heatmap of 6003 cells. This is not sensitive to caching because it is not cached. The fourth row is the time range for the temporal histogram: 53 weeks. The performance of this and the sensitivity to caching depends on the presence of filters. It can be much slower for queries lacking a keyword filter.

2. Components

2.1. Harvesting/Archiving

Harvesting is the process of capturing real-time data and loading it to the BOP. A harvesting service continuously polls Twitter's API based on predefined users and coordinate extents, then deposits the resulting JSON to a Kafka topic after serializing the tweet JSON using MessagePack. The latest billion geo-tweets are then pushed to Solr and made available for fast search and exploration via the BOP API (API 2017). CGA has been harvesting tweets since 2012 and has gathered about 8 billion objects to date.

2.1.1. Removal of Selected Bots

A twitter bot is a software program that sends out automated posts with auto-generated (often seemingly random) geospatial coordinates via Twitter. These geo-tweets, usually spread arbitrarily across the globe, create static in the geo-visualization without providing much benefit. We remove such bots at harvest time by user name, as we identify them. We document the bots we have removed so users can still retrieve them if needed directly from the Twitter API.

2.2. Enrichment

Enrichment consists of binary sentiment analysis (e.g. happy/sad) and geo-enrichment, e.g. census code, by reverse-geocoding against census and other boundary types. Both enrichment types will be discussed in detail below. As seen in Figure 2, this service consumes an input Kafka topic, and then adds the enriched tweets to another topic.

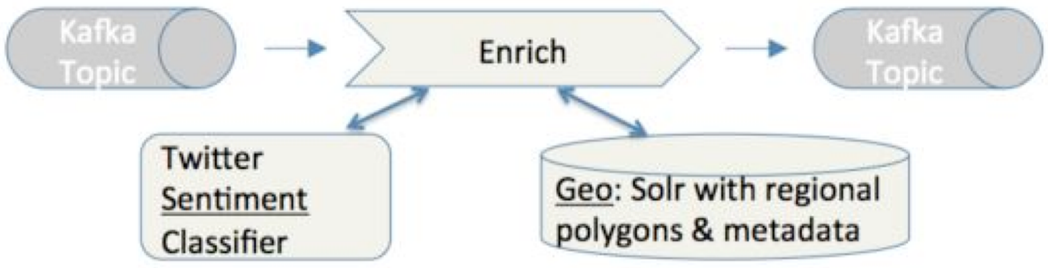

Kafka Topic
Enrich
Kafka Topic
Twitter Sentiment Classifier
Geo: Solr with regional polygons & metadata

Figure 2: Process of enrichment in BOP.

2.2.1. Sentiment Analysis

Sentiment analysis is a field of study which attempts to identify the emotion, expressed in text using Natural Language Processing (NLP) tools. Social media platforms such as Twitter provide a constant source of textual data, much of which is expressed with an emotion, which can be determined using Sentiment Analysis tools. We are using the Support Vector Machine (SVM) algorithm with a linear kernel, which is written in python and uses the scikit-learn library. Two classes of sentiment are assigned: positive (1) and negative (0). Tweets are pre-processed before sentiment stamping. The training corpi used are Stanford's Sentiment140, Polarity dataset v2.0, and the University of Michigan's SI650-sentiment classification dataset.

The process of sentiment analysis is divided in two phases: training and prediction as shown in Figure 3. In the Training phase we train the classifier and save it as a pickle file. In the prediction phase we load the trained classifier. Thereafter, for each tweet, we parse, pre- process, stem, and predict the sentiment. The precision, recall and F1-score of the algorithm is 82% for each. Sentiment enrichment takes an average of 20 milliseconds per tweet with no emoticon and 5 milliseconds for a tweet with an emoticon.

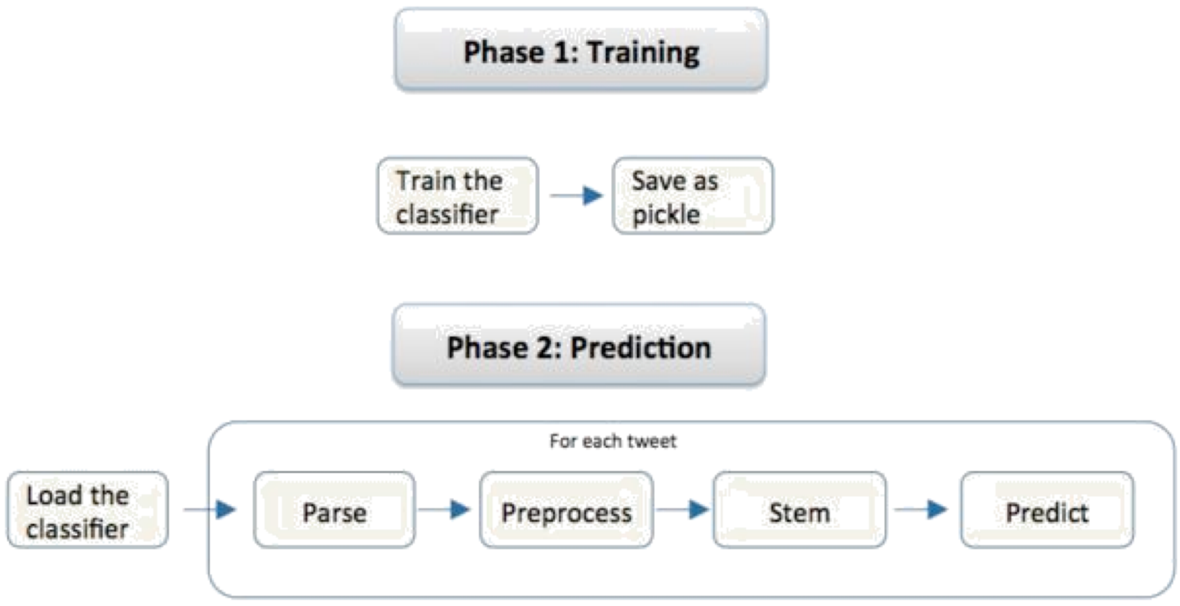


Figure 3: Sentiment analysis in BOP.

2.2.2. Solr for Geo-enrichment

Geo-enrichment uses Reverse Geocoding to search our polygon gazetteer using the tweet's geospatial coordinates to determine which census tract or county or country administrative polygon it falls within. An instance of Solr was loaded with these three data sets using separate Solr cores with a spatial field configured to support very fast point in polygon lookups. The total number of polygons used in our enrichment is about 250,000. The three queries for reverse geocoding enrichment only take about a millisecond each. The Solr field type contains some tuning details of interest:

```
"add-field-type":{
    "name":"geom",
    "class":
"solr.RptWithGeometrySpatialField",
    "spatialContextFactory":"JTS",
    "precisionModel":"floating_single",
    "validationRule":"repairBuffer0",
    "autoIndex":"true",
    "geo":true,
    "distanceUnits":"kilometers",
    "maxDistErr":0.005
}
```

There is also a geometry cache configured in solrconfig.xml as described in Solr's spatial documentation, which we configured to be of size 1024. Another performance factor is our use of EmbeddedSolrServer to embed Solr in the process.

2.3. Apache Kafka

Apache Kafka is a scalable message queue platform designed to handle high data volumes. In this project, the Kafka Streams API is used for consuming from a stream then passing the content to another stream and also for moving data from Kafka into other systems like Solr. Kafka does not support backpressure mechanisms, which creates challenges in some cases like using Kafka to re-index everything. A non- obvious use of Kafka we implemented is that of a persistent, long-term data store or archive. The archival tweets are stored in monthly partitions in a long-term Kafka topic. This approach for archiving has advantages in simplicity but also some serious limitations such as the fact data cannot be sorted or de-duplicated within Kafka as it could in a database. For the purpose of this project we set up a Kafka cluster consisting of 3 zookeepers and 3 brokers. There are 2 consumers for enrichment, an ingester, and 2 topics for input and output. The input topic contains real-time tweets and is fed by the harvester. The tweets from the input topic are consumed by the enrichment and fed to the output topic. Finally, the enriched tweets are consumed from the output topic by the BOP ingester, which transforms the data to a Solr document and then sends it to Solr. Yahoo's Kafka Manager API is used to manage Kafka.

2.4. Solr and Time Sharding

The BOP requirements are a perfect match for date/time-based sharding instead of the default hash based sharding. Sharding refers to the architectural organization of documents into logical slices of the entire dataset such that they can be accessed in parallel. Since Solr has no built-in date sharding mechanisms, a Solr plugin was developed which may be included within Solr in the future. The date sharding plugin is here: https://github.com/cga-harvard/hhypermap-bop/tree/master/bop-core/solr-plugins.

There are two components to the plugin. The primary one is DateShardingURPFactory which examines each tweet as it comes in, and then explicitly sets the special _route_ field based on the tweet's timestamp so that Solr's Distributed URP knows where to send it. The URP will create and remove Solr shards as needed while a tweet is being processed. The other component is DateShardRoutingSearchHandler, which is used instead of the conventional SearchHandler. This piece accepts additional search parameters sent by the client to establish the date range of the search query. The shards parameter is then generated based on applicable dated shards.

2.5. Enhancements to core Solr and Lucene

Apache Solr is an enterprise search server based on Apache Lucene. The following enhancements were made to Solr for the purpose of this project:

2.5.1. LatLonPointSpatialField

LatLonPointSpatialField (LLPSF) is a Solr FieldType that uses both LatLonPoint and LatLonDocValuesField – spatial components in Lucene. Together they are the state of the art spatial fields in Lucene and have shown to be the most performant for searching and sorting point data. LLPSF was released in Solr 6.5: https://issues.apache.org/jira/browse/SOLR- 10039

2.5.2. HeatmapSpatialField

HeatmapSpatialField (HSF) is a Solr FieldType that is expressly designed for heatmaps, unlike the existing RPT type in which it was an after-thought. The core difference between HSF and RPT is that HSF organizes the grid cells by resolution level together (breadth-first tree order), whereas RPT uses a depth-first tree order. High-resolution heatmaps will visit many same-resolution cells and so the intention with HSF is to increase data locality and use simpler cell iteration instead of seeking the Lucene term dictionary to each one.

After deploying HeatmapSpatialField and doing some basic performance measurements, the payoff over RPT was typically negligible in common queries, and there was only a 20% speedup for atypical queries that were already very fast. In essence, the high document count 10M+ per index was such that most time was spent iterating the postings lists (document ID lists) per term (cell), and not so much seeking to each term, despite there being many cells to traverse. Consequently, we didn't feel it was worth contributing this back upstream to Solr. The code for HSF can be found here: https://github.com/dsmiley/lucene- solr/tree/heatmap_hcga.

There are some other differences between HSF and RPT. One is a square grid cell, which is more ideal for data to be displayed than rectangles. Granted this aspect becomes increasingly negligible at higher resolutions. Another is the ability to omit certain resolution levels from being materialized into the index, to save space if you know you won't need them. For example the first level of practical use is not a 4x4 grid but more likely 32x32. These changes may find their way back to Lucene/Solr but have not yet.

2.6. BOP Web-Service

A RESTful API called the BOP Webservice was developed to support our BOP client and potentially other clients. The web service supports keyword search, space and time-based faceting, and CSV export. Having our own API instead of using Solr directly (which would also be possible) makes it much easier for developers to use, and it adds security. The BOP API is based on leading API frameworks Swagger and Dropwizard, and is written in Kotlin, a Java VM based language. The web service supports query/constraints on the following q.* parameters that limit the matching documents:

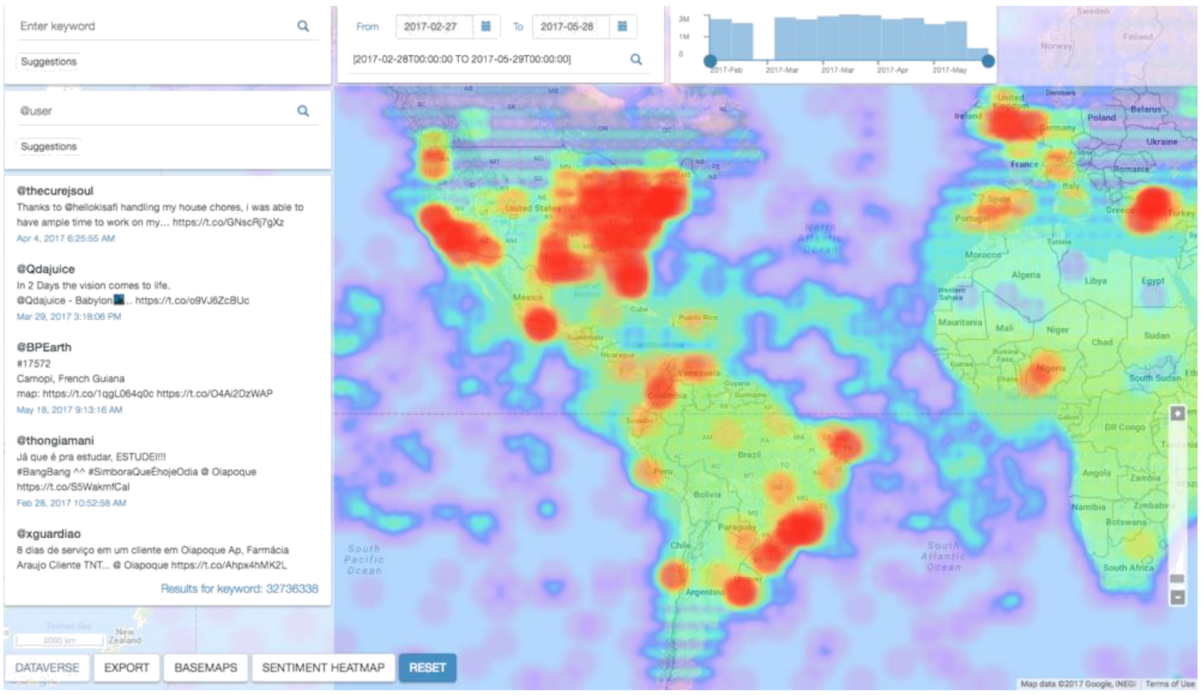
- q.time: Constraints docs by time range. Either side can be '*' to signify open-ended. Otherwise it must be in either format as given in the example. UTC time zone is implied.
- q.geo: A rectangular geospatial filter in decimal degrees going from the lower-left to the upper-right. The coordinates are in lat,lon format.

- q.text: Constrains docs by keyword search query.
- q.user: Constraints docs by matching exactly a certain user
- d.docs.limit: Limits how many documents to return.

Documents come back sorted by time descending. The response format is text/csv, comma separated with a header row. Values are enclosed in double-quotes (") if it contains a double-quote, comma, or newline.

2.7. BOP Client

The BOP client UI shown in Figure 4 is a browser-based UI with no server component, which makes HTTP/REST requests to the BOP Webservice. HTTP requests are proxied through a Kontena loadbalancer to provide access to the Kontena "weave" network which is one of the first overlay network technologies made available for Docker (Smiley and Kakkar 2017).



[2017-02-28T00:00:00 TO 2017-05-29T00:00:00]

@thecurejsoul
Thanks to @hellokisafi handling my house chores, i was able to have ample time to work on my... https://t.co/GNscRj7gXz
Apr 4, 2017 6:25:55 AM

@Qdajuice
In 2 Days the vision comes to life.
@Qdajuice - Babylon🌃... https://t.co/o9VJ6ZcBUc
Mar 29, 2017 3:18:06 PM

@BPEarth
#17572
Camopi, French Guiana
map: https://t.co/1qgL064q0c https://t.co/O4Ai2DzWAP
May 18, 2017 9:13:16 AM

@thongiamani
Já que é pra estudar, ESTUDE!!!!
#BangBang ^^ #SimboraQueÊhojeOdia @ Olapoque
https://t.co/S5WakmfCal
Feb 28, 2017 10:52:58 AM

@xguardiao
8 dias de serviço em um cliente em Oiapoque Ap, Farmácia Araujo Cliente TNT... @ Oiapoque https://t.co/Ahpx4hMK2L

Results for keyword: 32736338

Figure 4: BOP Client UI.

The client UI is built using Angular JS, OpenLayers3, and Node Package Module (NPM). The client UI can adapt to laptop, tablets and phones, and offers the following functionalities:

- Temporal filtering
- Temporal faceting (Histogram)
- Spatial filtering
- Spatial faceting (Heatmaps)
- Text faceting (Tag cloud)

The BOP web service sends out a matrix of data with tweet counts for the whole viewport. OpenLayers 3 with a custom algorithm is used to render that matrix into a heatmap. Rendering uses the Hue Saturation Luminosity (HSL) color model; the different tweet densities affect only the Hue but leaves Saturation and Luminosity unchanged. This helps make sure there are only vibrant colors instead of washed out color when doing the automatic interpolation. In order to be able to spot high-density areas, but also distinguish single tweets in sparse areas we used a sigmoid function to reduce the number of values in the middle classes.

The BOP client UI (UI 2017) allows for downloads of 10,000 tweet IDs at a time along with the enrichment fields in a CSV format. We have developed a tool in Python for rehydration of tweet IDs to fetch full tweet JSON using Twitter REST API's GET statuses/lookup method: https://dev.twitter.com/rest/reference/get/statuses/lookup. The tool returns fully hydrated tweet objects for up to 100 tweets per request, as specified by comma-separated values passed to an ID parameter. This method is useful to get details (hydrate) a collection of tweets. The order of tweets IDS may not match the order of tweets in the returned array. If a requested tweet is unknown or deleted, then that tweet will not be returned in the results list, unless the map parameter is set to true, in which case it will be returned with a value of null. If none of the lookup criteria matches valid tweet IDs an empty array will be returned for map set as false. One must be following a protected user to be able to see their most recent tweets otherwise their status will be removed. Twitter's POST method is preferred for larger requests.

3. Deployment/Operation

The BOP system is hosted on servers run by Massachusetts Open Cloud (MOC), which is a public cloud, based on the Open Cloud eXchange model and is built using OpenStack software.

The BOP runs on 12 virtual server instances: 5 for Solr, 3 for Kafka, 3 for enrichment, 1 as a proxy/gateway. In total it uses about 217 GB RAM and 3500 GB of disk space. Docker/Kontena metrics total 17 services and 133 Docker container instances.

3.1. Docker/Kontena

All components of the BOP are deployed to a Docker based infrastructure managed by Kontena. Docker is a leading software container platform, which provides ease of use: no installation, simplified configurations using environment variables and a common logging mechanism. It is ideal for deploying continuous integrated servers, it is still in its early days and the project requires some risk tolerance to use it in production, which was possible in our case with BOP being a prototype. Kontena was used to deploy Docker, and provides common logging, machine/process statistics, and security. Security is provided at the network or proxy level and is not service specific. Furthermore, Kontena VPN enables the creation of secure networks composed of servers in many locations with local access to all.

3.2. Admin Tools

We used the Yahoo Kafka Manager and the Solr Admin UI to manage Kafka and Solr respectively. The Kafka Manager provides support to manage clusters, topic, partitions and replicas. It also allows one to optionally enable Java Management Extension (JMX) polling for broker and topic level metrics and to filter out consumers. The Solr Admin UI is a web interface that makes it easy to view Solr configuration details, run queries and analyze document fields in order to fine-tune a Solr configuration.

4. Conclusions

Most mapping systems have difficulty representing millions of features on a map unless the visualization has been previously generated. This is because the system either attempts to render an image of millions of features to send to the browser, or it tries to send the features themselves to the browser as vectors. Both approaches are problematic. Rendering images that display millions of features is slow even on a fast server and this work is hard to distribute. Sending large numbers of features to the browser is also not ineffective.

Our approach takes a different tack, one, which attempts to optimize the characterization of the results. The solution is developed around the faceting capability of Lucene/Solr, which is traditionally used for topic counts. This project adapted facet in Solr/Lucene to support geographic counts (Faceting 2017). The approach enables a BOP server to return a spatially referenced array of result counts on the fly very quickly, which the browser then can render as a spatial surface.

The performance of this approach is quite impressive. Heatmap arrays based on queries against hundreds of millions of features are returned within sub-seconds. Speed does vary based on the complexity of the query. We made a mistake in sizing Solr shards by equal time intervals that resulted in large shards before 2016 when geo-tweet volumes were higher. The fix for this will be capping shards by document count as well as time. For earlier years performance is several seconds slower but still impressive when compared to most other mapping systems.

In addition to fast spatial faceting, the BOP also facets on time that enables fast rendering of temporal histograms depicting distribution in time as well as space. In addition to fast results rendering this project developed sentiment analysis (20 ms for tweet with no emoticon and 5 ms for tweet with emoticon) and reverse geocoding (sub-millisecond per poly). It turns out that Solr/Lucene, which is known for blazing text search, is blazing fast for spatial operations as well.

4.1. Future Work

The BOP is a prototype platform. Subject to funding, there are a number of areas, which can be developed, including but are not limited to the following:

- Return arrays of numeric (sensor variables for example) in addition to count values
- A UI to take advantage of census reverse geocoding and faceting to enable fast, on-the-fly spatial analytics using census variable on big data
- Integration with a Spark-based spatial analysis platform such as GeoMesa.

References

API, B., 2017. Bop api. Online, online; accessed May 31, 2017.
URL http://bop.worldmap.harvard.edu/bopws/swagger

Faceting, D., 2017. 2d faceting. Online, online; accessed May 31, 2017.
URL https://cwiki.apache.org/confluence/display/solr/Spatial+Search

Lewis, B., 2016a. Building an open source, real-time, billion object spatio-temporal search platform. Massachusetts Open Cloud Workshop, Boston, MA, USA.

Lewis, B., 2016b. Hhypermap registry: A platform to enable geospatial search. FOSS4G, Raleigh, NC.

OpenStack, 2017. Openstack. Online, online; accessed May 31, 2017.
URL https://www.openstack.org/

Smiley, D., 2016. H-hypermap: Heatmap analytics at scale. Lucene/Solr Revolution, Boston, MA, USA.

Smiley, D., Kakkar, D., 2017. Billion object platform (bop)- an open source, real-time, billion object spatio-temporal search platform. Harvard ABCD-GIS, Boston, MA, USA.

UI, C., 2017. Client ui. Online, online; accessed May 31, 2017.
URL http://bop.worldmap.harvard.edu/bop/