Scientific Research Publishing

Journal of Computer and Communications, 2021, 9, 97-109
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Reliable and Efficient Long-Term Social Media Monitoring

Jian Cao[1], Nicholas Adams-Cohen[2], R. Michael Alvarez[1]

[1]Division of the Humanities and Social Sciences, California Institute of Technology, Pasadena, California, USA
[2]Accenture, San Jose, California, USA
Email: jccit@caltech.edu, nicholas.adams-cohen@accenture.com, rma@caltech.edu

How to cite this paper: Cao, J., Adams-Cohen, N. and Alvarez, R.M. (2021) Reliable and Efficient Long-Term Social Media Monitoring. *Journal of Computer and Communications*, **9**, 97-109.
https://doi.org/10.4236/jcc.2021.910006

Received: September 7, 2021
Accepted: October 16, 2021
Published: October 19, 2021

Copyright © 2021 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

Social media data is now widely used by many academic researchers. However, long-term social media data collection projects, which most typically involve collecting data from public-use APIs, often encounter issues when relying on local area network servers (LANs) to collect high-volume streaming social media data over long periods of time. In this paper, we present a cloud-based data collection, pre-processing, and archiving infrastructure, and argue that this system mitigates or resolves the problems most typically encountered when running social media data collection projects on LANs at minimal cloud-computing costs. We show how this approach works in different cloud computing architectures, and how to adapt the method to collect streaming data from other social media platforms. The contribution of our research lies in the development of methodologies that researchers can use to monitor and analyze phenomena including how public opinion and public discourse change in response to events, monitoring the evolution and change of misinformation campaigns, and studying how organizations and entities change how they present and frame information online.

Keywords

Social Media, Cloud Computing, Twitter, Time Series

1. Introduction

Social media data is now widely used in many studies in computer and social science [1] [2]. Many of these studies collect short-term cross-sectional samplings of social media data, while others take advantage of free-access or paid social media APIs and attempt to build longer-term time series that monitor discussions and behavior online. For example, social scientists have been using shorter-term collections of social media data to study how political and social protests arise, for example in the case of the Arab Spring [3] [4] and Charlie Hebdo protests [5].

However, in any project that involves longer-term social media data collection efforts, there are many potential issues with collecting a reliable and consistent pipeline of social media data when using local area network servers (LANs). In this paper, we begin by discussing some of the issues that we have encountered in our own experience running a long-term Twitter data collection project (which at this point has been ongoing since 2014). We then present a cloud-based data collection, pre-processing, and archiving infrastructure which we argue mitigates or resolves many of the problems we have encountered, at minimal cloud-computing costs. We briefly discuss how this system could be applied to other social media platforms, and then explain specific applications of this methodology in the context of active research projects. Finally, we discuss how this architecture could be applied and enhance other research projects.

2. Problems Collecting Streaming Social Media

If a researcher is interested in quickly collecting cross-sectional social media data from Twitter, the use of the so-called "streaming" and "REST" APIs are relatively straightforward [2] [6]. Subject to rate limits, Twitter allows researchers to get access to a great deal of incoming Twitter data, including the content of a message, associated metadata, and information about the user account. In our application, where we study online conversations concerning political and social topics, collecting data from the Twitter Streaming API by keyword or hashtag data filtering over a brief window of time is relatively straightforward, and is a methodology that many scholars use in their research [7] [8] [9] [10] [11].

This situation becomes more complicated if the research project involves longer-term monitoring of conversations and discussions on Twitter. For example, one of our ongoing projects involves monitoring Twitter mentions of voter issues during elections, requiring us to collect data continuously in the weeks before, during, and after an election. Ever since beginning this project in 2014, we have worked to refine and improve our methodology for collecting these data [12] [13]. Our process focuses on searching for specific keywords that are associated with topics including election fraud, voting by mail, and registering to vote. In another example of long-term social media monitoring, we are developing methods for collecting Twitter conversations using dynamic keyword selection in situations where the discussion might be rapidly evolving over long periods of time [14].

In attempting to build long-term, multi-year, social media data collection projects on local machines, several prominent problems emerge. Our early work used Python scripts running on local university servers, connected to local-area networks. We found these scripts often encountered problems accessing the Twitter APIs, had trouble with network access, or competed with other processes running on the servers. In certain use cases, a good programmer is able to write test scripts to identify some of these issues, and pause collection to revise code [12].

However, even great programmers will have trouble resolving systems failures. An ideal social media monitor should maximize the amount of data gathered while minimizing the influence of any interruptions. Although a robust script that integrates diagnostic tools can help sustain the program, without solving the underlying system failures and limitations the threat of substantial data loss remains.

First, relying on local hardware introduces difficult, and in some cases impossible-to-anticipate, system failures. Power outages can knock systems offline, and without a secondary local system in place, data in a time series will be permanently lost. Network instability can also undermine data collection efforts, especially during peak-use hours or if network infrastructure is temporarily down for maintenance. Furthermore, if there is permanent system damage to a local system, it can be difficult, if not impossible, to recover data.

Second, the kinds of local systems most researchers have access to are not designed for the specific needs of collecting real-time streaming data. Collecting these data requires a system that can quickly and effectively obtain, buffer, process, and store large continuous streams of incoming information. Collecting this type of high-frequency, continuously streaming data involves specific computational considerations, with specialized algorithms designed to best capture these data [15]. Without a good system designed for these tasks, processing and saving files can temporarily interrupt the Twitter stream and limit the amount of data gathered. Slow processing that doesn't meet the publishing speed will also be constantly disconnected from the social media streams. Given these interruptions are most likely to occur during periods of heavy Twitter traffic, the censored Twitter data may not be missing at random, with systemic omitted Twitter data from the streaming API potentially biasing the results of a study [16].

Finally, setting up a LAN can potentially limit future collection efforts. As a collection project naturally expands, computational power, active memory, and storage considerations may change. However, local systems can be difficult, expensive, and time-intensive to upgrade, especially if one needs to address these concerns repeatedly in a multi-year research effort. On the other hand, for seasonal projects such as election monitoring, local systems can be less efficient as they are difficult to downgrade or temporarily shut down to cut expenses.

3. Cloud-Based Social Media Monitoring

In this paper, we present specific solutions to data collection on three popular cloud computing services: Google Cloud Platform (GCP), Amazon Web Services (AWS), and Oracle Cloud. While we illustrate our solution on these three platforms, the methods and processes we outline can be generally applied to other cloud services. While we do not claim here that we are the first to develop this type of workflow, we want to provide details about how our long-term social media data collection solution operates for other researchers to evaluate and utilize in their own work. Essentially, we built a process that solves or mitigates many of the problems using LANs by moving the data collection and pre-processing steps onto cloud-computing platforms.[1]

In the next sections of this paper, we first describe the general workflow of our social media monitoring system before providing detailed guidelines on how the process works in GCP, AWS, and Oracle Cloud.

3.1. Workflow

We developed an architecture using cloud resources to tackle the problems discussed in the previous section. This system collects as much social media data as rate limits allow in a stable and failure-tolerant manner.

The system consists of four parts: a data producer, a data stream, a data consumer, and storage. As shown in **Figure 1**, it starts with a producer that requests social media data from the API and acts as a data provider to the other parts of the system. Then the produced data are published in the data stream for temporary storage. The data stream secures every published record on a timeline according to the timestamp. Before the records expire, a data consumer retrieves them from the data stream and either sends them to analytics modules or to the short-term/long-term storage solution.

We describe the workflow of Twitter monitoring step-by-step in the following subsections and will expand the discussion to other social media platforms in the next section.

3.2. Data Producer

The data producer runs Python scripts on the cloud compute instances and connects two parts of our data pipeline: the Twitter Stream API and the cloud data stream.

The data producer:

- Uses the TwitterAPI package to access the Twitter Stream API, *google-cloud* to interact with GCP, *boto3* to interact with AWS, and *oci* to interact with Oracle Cloud.
- Connects to the Twitter Stream API using Twitter Developer credentials.
- Connects to the cloud data stream using credentials/tokens.
- Requests tweets from the Twitter Stream API.

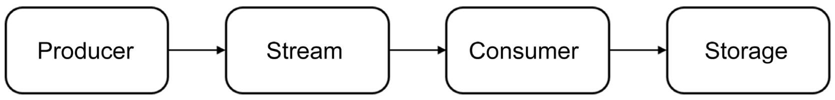

Figure 1. Social media monitor workflow.

[1]To reduce cloud-computing data storage costs, and to make the social media data we collect more readily accessible to our research group, we outline a process of piping the pre-processed data to cheap, secure, and easy-to-use data storage applications (here Google Drive). Note that, budget allowing, all data can be stored on a single platform.

- Publishes streaming tweets one-by-one to the cloud data stream.

We use the Twitter Stream API to collect real-time tweets. The most commonly used Twitter APIs for collecting tweets are the Stream and REST APIs. The Twitter Stream API delivers real-time tweets continuously as soon as they are posted and prior to subsequent alterations, such as deletion or censoring by the platform. Furthermore, as tweets are collected right after they are posted, they do not carry information regarding the replies and quotes that occur after a message is sent. On the contrary, the Twitter REST API allows users to search for the tweets that were posted in the past 7 days (30-day and full-archive endpoints are available upon upgrade). These tweets contain the latest retweet and quote information up to the time of extraction. However, instead of delivering all tweets that meet the filtering rules, the REST API only returns a subset of them that are most relevant. Therefore, to collect maximal tweets, our system focuses on the Stream API, only relying on the REST API as a backup in the case of interruptions.

It is important to note that the rate limits of the Twitter Stream API can cause potential data loss. The Twitter Stream API has a rate limit that only allows the delivery of up to 50 tweets/second. Some projects such as COVID-19 have exclusive unlimited endpoints. Those unlimited endpoints should be prioritized as they have no rate limits.

Selecting a suitable cloud compute instance is also important, as it ensures sufficient computing power at the lowest price. In our COVID-19 project, we use two data producers to request tweets of independent topics. We found that a Python script that produces at Twitter's rate limit (50 tweets/second) occupies on average 20% of a 2 GHz CPU core, and running data producers on separate CPU cores can prevent the processes from competing with each other for resources. For a two-producer project like ours, we suggest using an instance with two CPU cores. We further recommend allocating more than 1 GB of active memory space to compile large packages and run data consumer scripts. Table 1 shows three popular compute instances in GCP, AWS, and Oracle Cloud that

Table 1. Comparison of instance specifications.

Specs	GCP Compute VM n2-custom	AWS EC2 EC2 t3.medium	Oracle Compute VM.Standard.E2.2
CPU Series **CPU Model**	Intel (R) Xeon (R) Gold 6242	Intel (R) Xeon (R) Platinum 8175M	AMD EPYC 7551
Core Clock	2.80 GHz	2.50 GHz	2.00 GHz
Core Count	2 Cores	2 Cores	2 Cores
Memory	4 GB	4 GB	15 GB
Baseline Performance (per CPU)*	100%	20%	100%
Pricing	$50/Month	$30/Month	$61/Month

*CPU performance is restricted if average CPU usage exceeds the baseline.

each has specifications that meet the base requirements of the two-producer project. While the AWS system costs less than the others, AWS employs a baseline performance mechanism[2] that limits the instance's performance once the average CPU usage exceeds the baseline. For the t3.medium instance, performance will be significantly reduced if the user constantly uses more than 20% of the CPU(s). In general, the performance/price ratios are similar across platforms. Users can always customize or resize their instances to balance the performance and cost.

3.3. Data Stream

Cloud data streams are temporary storage services designed specifically for streaming data. Their role in the data collection pipeline is like librarians – they receive, organize, and preserve new information from the source and provide multiple means for the users to access the collected information. Data streams are the most essential part of the social media monitor, as they handle heavy data traffic that is beyond most local systems' capabilities, and make each streaming record available to all parts of the system. Data streams improve the reliability of the whole process by making it robust to failures caused by peaks of incoming data and serving as buffers for the subsequent parts of the workflow to re-visit data records if an error should occur.

While the data stream services have different names and specifications in GCP, AWS, and Oracle Cloud, they serve the same purpose and work in the same way. In the following discussion, we talk about the rate limits, retention periods, and pricing methods of these stream services.

Table 2 shows the default rate limits for the stream services Pub/Sub (GCP), Kinesis (AWS), and Oracle Stream. In the three services, Kinesis and Oracle Stream operate on basic units (called shards in Kinesis, and partitions in Oracle Stream), while Pub/Sub runs as a whole stream.

Table 2. Comparison of cloud data streams.*

Specs		GCP Pub/Sub	AWS Kinesis	Oracle Stream
Unit Limits	Write	None	1 MB/s 1000 Records/s	1 MB/s Unlimited Writes
	Read		2 MB/s 5 Reads/s	2 MB/s 5 Reads/s
Project Limits	Write	50 MB/s (small**) 200 MB/s (large)	Unit Limit × N	Unit Limit × N
	Read	100 MB/s (small) 400 MB/s (large)	Unit Limit × N	Unit Limit × N
Retention Period		7 Days	24 Hours	24 Hours
Pricing		Message Ingestion, Delivery, and Storage	Shards × Hour	Partitions × Hour

*Default rate limits can be increased upon upgrade. **Large regions: europe-west1, us-central1, us-east1. Small regions: other regions.

[2]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html

The basic units are pre-set components of a stream service. They are restricted by default rate limits and are independent of each other. Inside these units, published records are ordered by the timestamps of the "put" events. A shard of the Kinesis stream supports up to 1000 records/s or 1 MB/s (whichever is met first) for writes, and 5 requests/s or 2 MB/s for reads. For Oracle Cloud, a partition has similar rate limits to a Kinesis shard, except it has no restrictions on the number of write requests per second.

Users can create more units in their stream services to meet increased demand. In AWS Kinesis, a Python script can use metrics from AWS CloudWatch to monitor the usage of Kinesis shards, and scale up the number of shards if thresholds are met. For example, to prevent the incoming tweets from hitting the writing rate limits and causing data loss, we can use upper bounds 800 KB/s and 800 records/s as signals for immediate shard-creation. Once the average incoming tweets exceed one of the thresholds, the Python script immediately creates a new shard and lowers the average burden. We can also set lower bounds 500 KB/s and 500 records/s as signals for delayed shard-deletion. Once the average records per shard fall below the lower bounds, we want to delete one or more shards to reduce cost. To be conservative in deleting shards, we recommend waiting at least three hours after the lower bounds signals were triggered. The records that were published in the deleted shards will be available until expiration. For Oracle Cloud, it is not possible to add new partitions to existing streams, so users need to plan ahead or manually migrate to a new stream that has more partitions if incoming data surges.

The GCP Pub/Sub does not involve stream units explicitly. Instead of focusing on unit limits, Pub/Sub users must pay attention to project and resource limits. For a project that is located in large regions (europe-west1, us-central1, us-east1), all Pub/Sub topics combined cannot exceed 200 MB/s for writes and 400 MB/s for reads. The corresponding rate limits in other regions are 50 MB/s and 100 MB/s. The users should create separate GCP projects for large social media monitors that can potentially exceed the project limits.

The retention period determines how long a record is available to be read after being published in the stream. The default retention periods for Kinesis and Oracle Stream are both 24 hours, and it is 7 days for Pub/Sub. Users can upgrade to a longer retention period when creating the stream service.

There are two pricing methods, fixed pricing per unit × hour and flexible pricing per volume of data transmission and storage. Kinesis and Oracle Stream use the former and Pub/Sub uses the latter. The fixed pricing method leaves it to users to optimize the size and cost of the stream, and there are inevitable resource losses in running the units below the rate limits and in re-sizing the stream. On the contrary, flexible pricing only charges the resources being used. However, we wish to emphasize that flexible pricing does not necessarily lead to a lower overall cost.[3]

[3]Please refer to the pricing pages and the cost calculators for each cloud service to better estimate prices.

3.4. Data Consumer

Data consumers are cloud services or customized programs that read records from the data streams and send them to either cloud/local databases for storage or to analytical modules for data processing and analyses. In our social media monitors, we use DataFlow in GCP, Firehose in AWS, and the "get_messages" function in Oracle Cloud to extract records from the data stream. We prefer using a cloud service like DataFlow and Firehose instead of relying on customized programs given these services tend to have lower latency and higher stability. To add one more layer of safety, we send the extracted data immediately to cloud databases for short-term storage and subsequently invoke BigQuery (GCP), Lambda Functions (AWS), or Data Analytics (Oracle Cloud) for analyses.

3.5. Storage

Our social media monitoring system puts collected data temporarily in cloud storage before archiving data in Google Drive folders. Cloud storage services, such as Cloud Storage (GCP), S3 (AWS), Object Storage (Oracle Cloud) are natural data transfer and storage solutions between stream services and compute instances. They are ideal for frequently used data but are not cost-efficient for large sets of raw social media data collected over a long period of time. After the data collection pipeline is checked and the real-time analyses are completed, we store the data summaries and results of the analyses in a cloud MariaDB database and transfer the raw data to a Google Drive folder shared with all members of the research team. This three-layer storage structure (cloud data stream–cloud storage-Google Drive) is robust to system failures given it inherits the stability and compatibility from the cloud services, and, if an error should happen, the three-layered system can easily recover data from a previous step.

4. Other Social Media Platforms

Of course, Twitter is not the only source of dynamic social media data for researchers. While the Twitter API's relative ease of use and open policy make it one of the most popular sources of data in academic studies, many researchers conduct research utilizing data from other popular social media platforms with APIs, such as Reddit [17] [18], YouTube [19] [20], and Facebook [21] [22].

Our approach can be adapted to collect streaming or dynamic social media data from these other platforms. Considering **Figure 1**, the main modification in our process would be in the **Producer** step of the workflow. In this step, we describe setting up a virtual compute instance capable of running code designed to interact with the Twitter API. To modify our process to collect other forms of streaming data, we would simply rewrite this code to interact with another platform's API.

For example, suppose we wished to collect data over time from a particular Reddit community (a subreddit). We could reuse the same process described in the previous section, revising the set of scripts in the Producer step to access data from the Reddit API. There might also be some minor changes required in the code used in the preprocessing and consumer steps, as the data stream from the Reddit API may differ from the Twitter API. These minor alterations aside, conceptually our approach is highly adaptable for collecting streaming and dynamic data from a variety of social media platforms, as long as they have a public API or their data can easily be obtained via a script that can run on a cloud computing instance.

5. Applications

5.1. Monitoring COVID-19 Tweets

Since January 2020, we deployed two Twitter monitors on GCP to collect tweets related to the COVID-19 pandemic. Both monitors employ the cloud architecture shown in Section 3. The first monitor (the "static" keyword monitor) was launched on January 31st, 2021, and was retired on June 30th, 2021. It used a VM n2-custom instance shown in **Table 1** as the data producer, a PubSub stream, and a DataFlow instance as the data consumer. While in operation, the monitor listened to the Twitter stream endpoint "*/statuses/filter*" and used a static keyword list to filter the real-time tweets. It collected around 1.86 billion tweets (8.72 TB in size) in 516 days with a peak ingestion rate of 150 tweets/second. The second monitor (the "dynamic" monitor) was launched on June 1st, 2020, and is still active at the time of article submission. The monitor listens to the Twitter COVID-19 lab endpoints "*labs/1/tweets/stream/covid19*", which generates tweets that are related to a wide range of COVID-19 pandemic topics. The keyword list is dynamic.[4]

It is managed by Twitter and contains more than 500 topics in multiple languages. The COVID-19 lab API has four endpoints and does not have a rate limit. Therefore, we deployed an instance with 4 cores to ensure that the endpoints do not compete CPU time. Since June 2020, the dynamic monitor collected around 5.7 billion tweets (29.04 TB in size) with a peak ingestion rate of 250 tweets/second.

We show the daily number of tweets collected by both monitors in **Figure 2**. The blue trend is the static monitor and the red one is the dynamic monitor. We observe a quiet start in February 2020. But discussion of the pandemic skyrocketed in March, reached a peak in April, and gradually slowed down in May. After the second monitor was launched in June 2020, because of the larger keyword list, the data collected increased significantly. However, the downward-sloping trend was not affected. Over a year since then, the daily number of tweets collected by the monitors keeps declining, but very slowly. The fluctuations can be observed every several months, which correspond to the waves of the pandemic. At the time of article submission, the COVID-19 lab monitor collects on average 5 million tweets per day.

With the help of the social media monitoring architecture, we are able to construct a comprehensive dataset of COVID-19 tweets. The extremely large volume

[4]https://developer.twitter.com/en/docs/labs/covid19-stream/filtering-rules

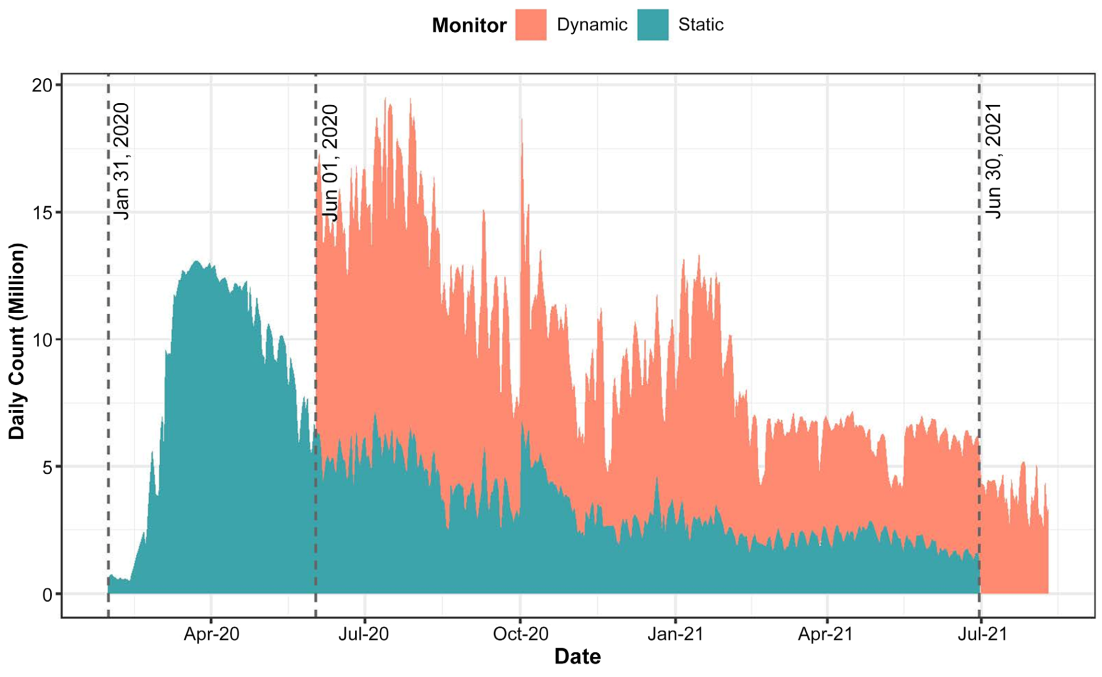

Figure 2. Daily tweet count of the COVID-19 monitors.

of collected information and high peak ingestion rates demonstrate how this dataset could only be reliably constructed and with a robust cloud architecture. Currently, our research group is using these data to study online misinformation, toxic speech, and how politicians have framed their discussion of government pandemic response during the COVID-19 pandemic.

5.2. Monitoring Misinformation and Rapidly Evolving Online Conversation

Our research group also uses this infrastructure to collect near real-time social media datasets that we use to detect misinformation and toxic speech. There are a number of reasons why having a robust, fast, and comprehensive system for the collection of large amounts of social media data for these scientific projects is essential. Misinformation and toxic online speech are of course harmful to the individuals who receive it or who it might be directed at, and as we have witnessed in recent elections worldwide, deliberate misinformation campaigns might be problematic for democratic elections. But, such speech in social media can be difficult to detect—it is low incidence activity, and those who engage in these activities have strong incentives to mask their behavior to avoid detection and removal from social media platforms [14] [23].

We have used this architecture to collect a large dataset of Twitter data during the 2020 presidential election, in particular a dataset of approximately 56 million tweets on election and voter fraud. During this election, we used the data produced by this architecture to power a number of real-time visualizations of the conversation online about many aspects of the election. These visualizations can be seen on our Monitoring The Election website, https://rmichaelalvarez.github.io/twitter-monitors.html. We then use these data for deeper forensic studies of geographic locations where there is a conversation about election problems [13]. This provides another example of the practical and scientific usefulness of our data collection architecture.

Also, in a recently published paper, our research group used this architecture to collect Twitter streaming data in the aftermath of the January 6, 2021 Capital riots, in particular, in the period leading up to and during President Biden's Inauguration [23]. At the time there were significant concerns that social media might be used to organize and facilitate efforts to disrupt the Inauguration, and our monitor was built to track an array of keywords associated with the Inauguration, but to also evolve that set of keywords as the conversation shifted between January 11, 2021, and January 22, 2021. As we show in our paper, while there was a significant change during this period in how Twitter users discussed the event, we did not see the evolution of significant levels of toxic conversations [23]. This is another example of how our social media data collection architecture can produce reliable and quick streams of large quantities of data, which can then be used in practical and scientific applications.

6. Discussion

Many research groups are using social media data in their studies of political, social, and economic attitudes and behavior. Interest is increasing in the development of longer-term datasets that can be used to analyze changes over time in attitudes and behavior [11]. However, in our efforts to collect longer-term social media datasets from local servers using local area networks, we have encountered important limitations in the availability and reliability of those systems for these purposes.

To improve the reliability of our longer-term social media data collections process, we have developed a cloud-based infrastructure that is adaptable to several platforms. By moving the data collection and pre-processing stages into the cloud, we avoid many of the problems encountered when relying on local servers and LANs. We designed our process to make data easily accessible to our research group by moving the data to a secure and usable storage solution like Google Drive. We have shown that these processes work across different cloud computing systems, and can be used to collect both Twitter and other streaming social media data.

7. Conclusion

There are many opportunities for researchers who are interested in using social media data to study the longer-term dynamics of political, social, and economic attitudes and behavior. We hope that by providing the details of our framework, other researchers can evaluate and potentially use this report as initial guidance in adopting a cloud-based process, improving their ability to collect similar datasets.

Acknowledgements

We thank the John Randolph Haynes and Dora Haynes Foundation for supporting some of this research. We also thank the Google Cloud Research Credits Program for providing cloud computing credits to support our research on COVID-19. We thank Anima Anandkumar and Anqi Liu for their collaboration and support of our research. The collected tweets and summary statistics are available upon request from the corresponding author (jccit@caltech.edu). Python scripts that are used to build the Twitter monitors are available in the Github repo https://github.com/jian-frank-cao/MonitoringTwitter.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Klasnja, M., Barbara, P., Beauchamp, N., Nagler, J. and Tucker, J. (2018) Measuring Public Opinion with Social Media Data. In: Atkeson, L.R. and Alvarez, R.M., Eds., *The Oxford Handbook of Polling and Survey Methods*, Oxford University Press, Oxford, 555-582. https://doi.org/10.1093/oxfordhb/9780190213299.013.3

[2] Steinert-Threlkeld, Z. (2018) Twitter as Data. Cambridge University Press, New York. https://doi.org/10.1017/9781108529327

[3] Barberá, P., Wang, N., Bonneau, R., Jost, J.T., Nagler, J., Tucker, J. and González-Bailón, S. (2015) The Critical Periphery in the Growth of Social Protests. *PLoS ONE*, **10**, e0143611. https://doi.org/10.1371/journal.pone.0143611

[4] Steinert-Threlkeld, Z.C. (2017) Spontaneous Collective Action: Peripheral Mobilization during the Arab Spring. *American Political Science Review*, **111**, 379-403. https://doi.org/10.1017/S0003055416000769

[5] Larson, J.M., Nagler, J., Ronen, J. and Tucker, J.A. (2019) Social Networks and Protest Participation: Evidence from 130 Million Twitter Users. *American Journal of Political Science*, **63**, 690-705. https://doi.org/10.1111/ajps.12436

[6] Russell, M.A. (2014) Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites. Second Edition, O'Reilly Media, Inc., Sebastopol.

[7] Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media (2010) From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series.

[8] Flammini, A., Conover, M.D., Gonçalves, B. and Menczer, F. (2012) Partisan Asymmetries in Online Political Activity. *EPJ Data Science*, **1**, 1-19. https://doi.org/10.1140/epjds6

[9] Barberá, P. and Rivero, G. (2014) Understanding the Political Representativeness of Twitter Users. *Social Science Computer Review*, **33**, 712-729. https://doi.org/10.1177/0894439314558836

[10] Beauchamp, N. (2017) Predicting and Interpolating State-Level Polls Using Twitter Textual Data. *American Journal of Political Science*, **61**, 490-503. https://doi.org/10.1111/ajps.12274

[11] Adams-Cohen, N. (2020) Policy Change and Public Opinion: Measuring Shifting Po-

litical Sentiment with Social Media Data. *American Politics Research*, **48**, 612-621. https://doi.org/10.1177/1532673X20920263

[12] Adams-Cohen, N.J., Hao, C., Jia, C., Matschke, N. and Alvarez, R.M. (2017) Election Monitoring Using Twitter. Technical Report Working Paper 129, Caltech/MIT Voting Technology Project.

[13] Alvarez, R.M., Adams-Cohen, N., Kim, S.Y.S. and Li, Y. (2020) Securing American Elections: How Data-Driven Election Monitoring Can Improve Our Democracy. Cambridge University Press, Cambridge. https://doi.org/10.1017/9781108887359

[14] Liu, A., Srikanth, M., Adams-Cohen, N., Alvarez, R.M. and Anandkumar, A. (2019) Finding Social Media Trolls: Dynamic Keyword Selection Methods for Rapidly Evolving Online Debates. *AI for Social Good Workshop*, *NeurIPS* 2019, Vancouver, 14 December 2019. https://arxiv.org/abs/1911.05332

[15] Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J. (2002) Models and Issues in Data Stream Systems. *SIGMOD/PODS*02: *International Conference on Management of Data and Symposium on Principles Database and Systems*, Madison, 3-6 June 2002, 1-16. https://doi.org/10.1145/543613.543615

[16] Morstatter, F., Pfeffer, J. and Liu, H. (2014) When Is It Biased? Assessing the Representativeness of Twitter's Streaming API. https://doi.org/10.1145/2567948.2576952

[17] Tan, C., Niculae, V., Danescu-Niculescu-Mizil, C. and Lee, L. (2016) Winning Arguments: Interaction Dynamics and Persuasion Strategies in Good-Faith Online Discussions. *Proceedings of the* 25*th International Conference on World Wide Web*, Montreal, 11-15 April 2016, 613-624. https://doi.org/10.1145/2872427.2883081

[18] Nithyanand, R., Schaffner, B. and Gill, P. (2017) Online Political Discourse in the Trump Era. Technical Report, ACM.

[19] Burgess, J. and Matamoros-Fernández, A. (2016) Mapping Sociocultural Controversies across Digital Media Platforms: One Week of #gamergate on Twitter, YouTube, and Tumblr. *Communications Research and Practice*, **2**, 79-96. https://doi.org/10.1080/22041451.2016.1155338

[20] Rieder, B., Matamoros-Fernández, A. and Coromina, Ó. (2016) From Ranking Algorithms to "Ranking Cultures": Investigating the Modulation of Visibility in YouTube Search Results. *Convergence*, **24**, 50-68. https://doi.org/10.1177/1354856517736982

[21] Rieder, B., Abdulla, R., Poell, T., Woltering, R. and Zack, L. (2015) Data Critique and Analytical Opportunities for Very Large Facebook Pages: Lessons Learned from Exploring "We are All Khaled Said". *Big Data & Society*, **2**, 1-22. https://doi.org/10.1177/2053951715614980

[22] Kalsnes, B. (2016) The Social Media Paradox Explained: Comparing Political Parties' Facebook Strategy versus Practice. *Social Media + Society*, **2**, 1-11. https://doi.org/10.1177/2056305116644616

[23] Srikanth, M., Liu, A., Adams-Cohen, N., Cao, J., Alvarez, R.M. and Anandkumar, A. (2021) Dynamic Social Media Monitoring for Fast-Evolving Online Discussions. *KDD*'21: *Proceedings of the* 27*th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Singapore, 14-18 August 2021, 3576-3584. https://doi.org/10.1145/3447548.3467171