DATA MINING I:

Utilizing Database Architecture and Data Cleaning to Increase 'Time to Science' and Decrease Resources Needed in Research Data Science Workflows

3 February 2020

Christine R. Kirkpatrick

Department of Computer Science, School of Sciences of University of Porto, Portugal *and*
San Diego Supercomputer Center, University of California San Diego, USA

Table of Contents

Utilizing Database Architecture to Decrease Resource Utilization in Research Data Science Workflows

Abstract

In this work, one research workflow was studied to pinpoint the greatest impediment for 'time to science' or the amount of time before the user (a researcher) could begin their work. The database ingestion portion of a research workflow was slow enough that researchers were limiting the data studied, rather than waiting for the full dataset requested to become available. This research workflow utilized data available via Twitter's API for several use cases from public health to computational social science. Several areas for potential speed up of record ingestion were studied, including the shared cloud resources and the data's characteristics. After examining shared file system load and looking for differences in the individual ingestion files, the presence of duplicate tweets emerged as a major complication. When the primary key was changed from the source record ID, in this case the tweet ID, to a unique auto-incrementing integer, ingestion time sped up 350%. Computing optimizations equate to savings in cost and energy usage. To ingest a year of the '1% feed' Twitter data, this time savings translates to $342.11 (€307.65) in cloud cost savings and 19.3 kW in energy savings or the equivalent of CO^2 emissions from charging a smartphone 1,740 times or burning 15 pounds (6.8 kg) of coal.

Background

Putting large-scale/big data into a relational database is time consuming. While other approaches exist for collecting and querying data, many researchers are not ready for the paradigm shift away from SQL and structured data. Ingestion into a relational database is resource (time) prohibitive and limits the time scientists have for analysis and inquiry. Researchers must wait for the data to be ready and often reframe research questions to require less data – or draw conclusions based on much smaller data sample sizes. An overarching goal of this research is to speed up researchers' 'time to science' or rather, decrease the amount of time spent getting data ready and compute environments configured before inquiry can begin. Decreasing the use of resources needed to accomplish the same analysis holds the potential to save researchers' limited funds, as well as to contribute to decreasing computing's climate impact.

Twitter as a Multipurpose Dataset for Many Research Domains

Social media datasets, such as can be obtained from Twitter using the free 1% API feed, provide data and computer scientists with a versatile resource for testing different techniques, such as clustering and topic modeling. Scientists and their lab staff are often most familiar with relational databases and SQL, preferring to use databases such as MySQL and PostgreSQL (Oracle 2020, PostgreSQL 2020). The same datasets can be converted for use in specific-purpose database platforms, such as Solr for text mining, and Neo4J for graphs (Solr 2020, Neo4J 2020).

Twitter data was selected for its usefulness in several domains. Political scientists query Twitter datasets to look for co-occurring keywords to detect relationships of concepts over time, as well as to build graphs to detect network development and influence (Zheng 2019, Steinert-Threlkeld 2019). Twitter is useful in public health to augment (public health) surveillance systems that may only release data annually; Twitter data can provide timely insight into aggregate behavioral trends such as unsafe sex or drug use (Benbow 2019, Guo 2019).

> Its high degree of spatial and temporal granularity allows the study of behavior at low levels of aggregation but also at a more macro scale and from a comparative perspective. The fact that human behavior is observed unobtrusively also facilitates collecting data at a larger scale and reduces certain types of biases (Barberá 2019).

Across projects requiring even short period samples (days) of social media data, the time to ingest or copy already available data into a database is the most time-consuming portion of the workflow and an impediment to its intended use by the data science or domain study that requires the data resource. The San Diego Supercomputer Center (SDSC) has been collecting tweets for its data scientists for various applications since 2018 and has over 15 billion tweets (records) in their native (file) format, stored in a cloud bucket (object store). At 700GB uncompressed, the tweets can't be loaded onto a single system using open source software. Despite having data available for analysis, it is a constant challenge to make the data available in a form that can be queried and is useful to researchers.

Related Work

Recent publications such as Oliviera examine how moving scientific workflows to the cloud bring complexity, especially as it relates to predicting failures. "The complexity of cloud infrastructures (i.e., the large number of hardware and software components and their interdependence relationships) raises the need for performance evaluation methods that consider the failure of components" (2019). Because of the complexity and cost and time constraints for testing many permutations and variations of configuration, state-based models are often used for performance evaluation, e.g. Stochastic petri nets. Gathering job characteristics such as "response time, throughput, availability" can be a path to finding appropriate cloud architectures (Andrade 2019). Where such literature shows a gap is in skipping optimization to focus on the performance and dependability of well-defined information technology (IT) loads. Much of the experimentation in the literature is making use of simulators, notably CloudSim, rather than experimentation based on real world tests or validated using worked examples (Oliviera 2019, Erradi 2020, CloudSim 2020). Further the focus on cost optimization is not always aligned with saving power or computing cycles. At the San Diego Supercomputer Center, compute-intensive racks, such as for high performance computing (HPC) average 12 kilowatt hours (kWh) per 48u rack versus the average of 4 kWh for non-HPC, and often far less for storage only racks (Filliez 2020). Storage can be the costliest element in a commercial cloud bill, which in turn can add complexity to a workflow, such as in trying to move data between archival (cheaper) storage and any kind of 'hot' or spinning disk storage, so that at any one moment, only the data being computed is on spinning disks (Erradi 2020).

There is a gap in the literature concerning cloud use for research data science loads. Much research has been done to optimize workloads where the parameters are unchanging, such as persistent virtual machines that host databases with very small add/delete/update queries. It is understandable that much attention is paid to industry loads because of the monetary incentive to save money and/or to speed up processes that will lead to increased revenue, e.g. earlier, more frequent product shipments (Beloglazov 2012). It is somewhat inconsistent to focus research on predictable loads given that cloud is inherently unpredictable due to its use of shared, commodity hardware (Homer 2014).

Concerning the particular workflow studied and its use of PostgreSQL: there are numerous works concerning PostgreSQL best practices and a large and active user community (Geschwinde 2002, Obe 2017, PostgreSQL 2020). However, none were found to address the optimization opportunities for users of shared infrastructure, i.e., who lack access to the underlying file system. Where the literature is helpful in a cloud context, is in regard to memory optimization for PostgreSQL. Borodin found memory improvements that tripled the speed to copying data into an index by three (2017). However, the specific recommendation was suggested as an improvement to PostgreSQL's code base and not something the user can control. Additional literature review and matching of database design and schemas should be done as suggested by additional workflows and bottlenecks related to interaction with database management systems (DBMS).

Specific Challenge Examined

The workflow analyzed begins with obtaining Twitter data from the Twitter API and ends with a populated PostgreSQL database ready for querying. With historical Twitter data available on the local cloud already, a common task is to create a new database with a sample of *x* months of tweets. Scientists would prefer more data, at least a year's worth, but at current processing speeds it was time prohibitive: taking over 28 days just to copy available data on the same 10GB network into a database - and often much longer because of duplicate records that would cause data ingestion to fail. Data ingestion was found to be one of the workflow's significant bottlenecks.

Description of the Workflow Studied

The workflow chosen for examination creates an open source relational database using data captured from a Twitter stream. This is accomplished in three distinct steps. Cloud instance sizes discussed refer to specific templates used to configure cloud resources. Table 1 summarizes relevant resource differences between instance types mentioned.

1. **Data Capture** - an *xe.large* instance utilizes Apache Heron, Twitter's open source streaming service for capturing large amounts of data. A Twitter key authenticates the API calls from Heron. Heron obtains the tweets from Twitter in JSON format, stores them on a RAM disk, and periodically writes the records to a compressed .zip file. The files are first stored locally on the cloud instance. Each hour a new file is created and the previous one is uploaded to OpenStack Swift via Swift's native protocol (similar to, but not S3) (Swift

2020). Each one hour file is approximately 200 MB and contains 2.5 million tweets (records).

2. **Data pre-processing** - a constant issue is the occurrence of duplicate records. In the current workflow, the tweet IDs are used as unique keys. The presence of duplicate records will cause the individual file to fail when it is attempted to copy a record with a duplicate primary key into the table. It was assumed that Heron was introducing the duplicate records, so a short Java program was written to capture the Twitter feed instead. This appears to have corrected the issue going forward, however, duplicates still remain in the large number of tweets collected already. The current workflow design transforms the JSON into CSV that matches the database schema (see Appendix I).
3. **Ingestion (DBMS Schema /copy/insert) -** The database schema used is from GNIP (see Appendix I) and combined two schemas: one for activities (tweets) and one for actors (users). This was compared against Twitter's object data dictionary, and "tweet" and "user" were later added. In the time since the data collection began, Twitter doubled the number of characters per tweet and added an additional field with the data above 140 characters. This made it necessary to combine the two text fields from newer, longer tweets into one field for easier analysis. Once the data is pre-processed, it is copied into PostgreSQL via a *copy* command (Appendix II). Each hour or ~2.5 million records are copied in a batch. A shell script controls the serial batching of data ingestion.
   It should be noted that in the original workflow, a custom schema was used that kept the tweets and users in two separate tables**.** The GNIP schema was employed for testing in an attempt to limit complications from user customizations.

Shared Resources and Cloud Computing Introduce New Complications for Diagnosing Bottlenecks

The workflow examined runs on *SDSC Cloud*, a private cloud based on OpenStack (SDSC Cloud). OpenStack is an open source software project that allows for hosting infrastructure as a service (IaaS) style compute and storage instances, similar but much more simplistic than Amazon Web Services (AWS) or Google Cloud Platform (GCP) (Sefraoui 2012). The relative simplicity of billing and limited menu of services can be welcome features for scientists doing grant-funded research who find commercial cloud's options and billing complexity vexing.

SDSC Cloud is a shared resource. Every attempt was made to isolate the processing times although the underlying storage and networking infrastructure is shared by the entire SDSC Cloud. The workflow utilizes three cloud instances, all with 20 GB of storage for the operating system (Table 1). The cloud instances use attached block storage managed by Ceph with 3 copy replication (Weil). Ceph is set up with a 5:1 ratio of spinning hard disks (HDDs) to solid state drives (SSDs). The tweets are stored on in object store using OpenStack Swift (Swift). The object store is configured to store three copies and can be accessed via an S3 API.

	Size	vCPUs	RAM	Storage
Heron Tweet collection	xe.large	2	64 GB (32 of 64 as a RAM disk)	S3 bucket
Tweet pre-processing	m1.medium	1	40GB	
Social media database	R.1xlarge	4	32 GB	1 TB block

Table 1: Cloud instances utilized in the workflow studied.

The database utilized is PostgreSQL (Stonebreaker 1991). It was chosen because it is open source, has a wide user base, and is relatively stable. Prior to PostgreSQL, Apache AsterixDB was used, an open source DBMS designed for big data needs (Grover 2015). While it excels at ingestion of real-time data streams, it was relatively new and undergoing frequent overhauls; features were not always stable between releases, leading to the need to reload data. As well, storage of the raw data was appealing, rather than ingestion straight into AsterixDB, as the data could be imported into other types of DBMS. It was decided to convert to a more stable DBMS that the research computing team had considerable experience using.

The schema used is a sample database schema provided by GNIP, the company owned by Twitter that manages the Twitter API and data dictionary (see *Appendix I* for schema details). As with many modern DBMS's, Postgres uses a B+- tree index which allows for flexibility in supporting different types of queries including equality and range (Stonebreaker 1991). Aside from database tuning, performance is related to the I/0, CPU, and the underlying file system (Geschwinde 2002). However, much of the advice in the literature is not applicable to cloud-based implementations as researchers do not gain access to the configuration of the shared file system. In today's IT structures where analysts specialize in systems management *or* database management (but rarely both), tuning a system for the DBMS requires careful cooperation and shared knowledge; the systems person should possess basic database knowledge, and database administrators (DBAs) should understand the system they rely on.

Memory is another key resource, which in a cloud environment can be controlled by the user. Cloud users can create RAM disks out of the memory allocated. For the workflow studied, a RAM disk was created for the cloud instance running the data import from Twitter. Of 64 GB of memory allocated to the instance, half (32GB) was used as a RAM disk that performed at roughly twice the speed of a regular spinning disk or hard (disk) drive (HDD). While a powerful technique for storing data before writing to a file, the RAM disk did not appear applicable to the bottleneck examined. Future study should include use of solid state drives (SSDs) in cloud architectures. SSDs are in use in the underlying cloud nodes and storage subsystems of SDSC Cloud. However, dedicated SSDs are not currently available to SDSC Cloud users. This is due to the limitation in allocation. SSDs must be allocated by the size of the physical resource and can't be shared. For example, a 1 TB SSD must be allocated as 1 TB, even if the cloud instance only needs 100GB or 10% of the available resource.

Experimentation

Suspected Causes of Data Ingestion Slowness and Failure

The data scientists using the system believed the ingestion was related to the size of the database and that more resources, such as added memory or CPUs, might speed up processing times: ***Hypothesis$_1$*** *- Ingestion time increases as the database grows in size.* Because of the issue with duplicate records causing the processing of individual files to hang or fail, this led to the supposition that locating and omitting duplicate records prior to ingestion might speed processing: ***Hypothesis$_2$*** *- Ingestion time can be decreased through data preprocessing.* Aside from the first two hypotheses, it seemed prudent to ensure that best practices were being followed for the specific DBMS: ***Hypothesis$_3$*** *- Ingestion can be sped up by implementing best practices related to record inserts and updates for the specific DBMS*.

Exploration and Experimentation to Speed Up Ingestion

Database Size in Records

Hypothesis$_1$, is that ingestion time increases as the database grows in size. Using the research workflow with no adjustments, the ingestion was re-run. Sample processing times for a database that already contains 33M records, as it grows to 100M records was analyzed and show to vary widely (Appendix III). Given the relative homogeneity of the individual files (in number of records

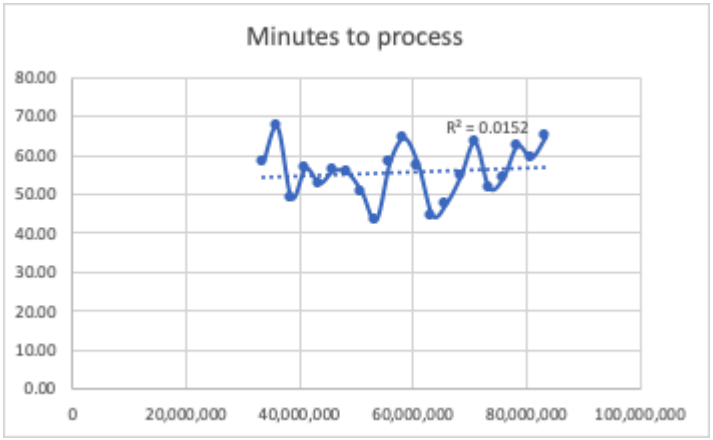

Figure 1: Sample ingestion times given database size in records.

and data type), it became unclear how the run times could range from 43 - 67 minutes for the same amount of data. It showed no correlation (R^2 of .015) between the number of records in the database and ingestion times (Figure 1).

Shared Resources and the File System

Shared infrastructure load was another area to investigate. This exposed that previous ingestion job logs were not capturing the timestamp for when the individual files were either started or completed, making it difficult to look for correlations between the processing times and shared

system load. The ingestion script was augmented to include timestamps per file processed going forward. The resource that seemed most variable and likely to impact performance based on the literature review is the shared file system, based on Ceph (Ceph 2020). Looking at samples of load in Ceph logs viewed via Graphite, a log visualization platform, there were no immediate patterns that correlated to the variability in ingestion times (Graphite 2020). This was set aside as a place to investigate further once the ingestion logs had timestamps to match with Ceph load levels.

Once the experiment with timestamped jobs was available, it showed no discernible pattern between processing times and Ceph load as expressed in reads or writes (by bytes per second) (Figure 2). Ceph logs were converted from PST to UTC to align with ingest job timestamps. Ceph

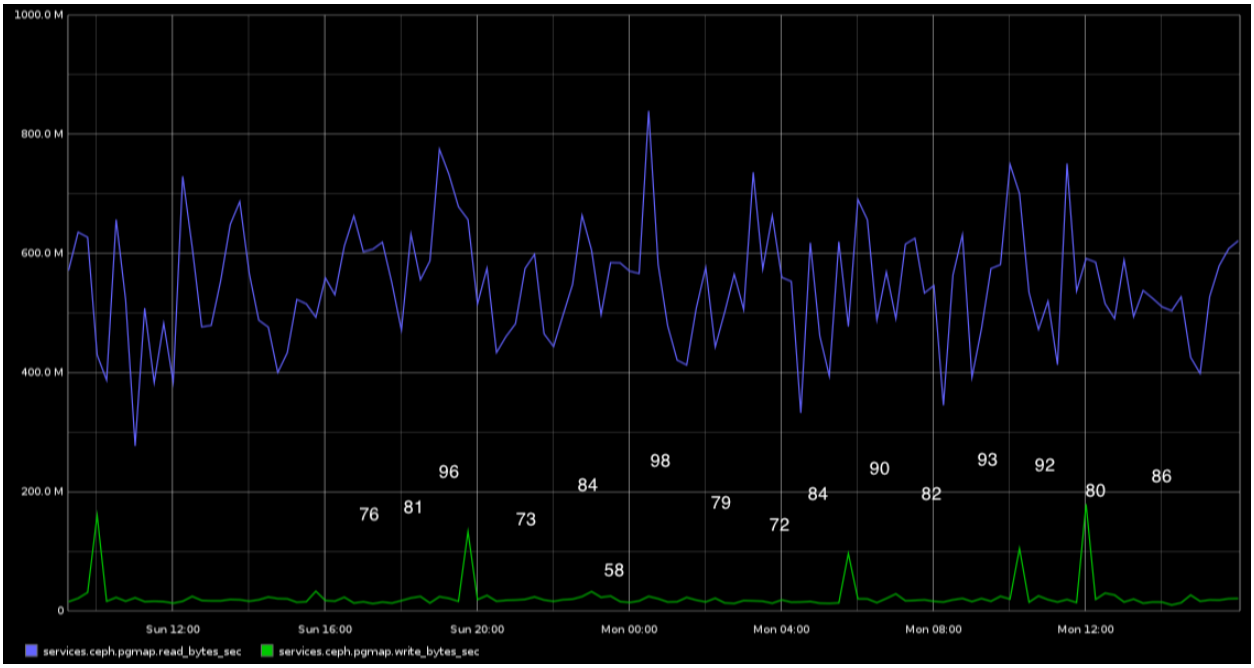

Figure 2: Sample of Ceph block storage activity. All OpenStack cloud instances share this storage resource. Top line shows reads, bottom line shows writes. Plotted numbers denote the end of jobs in minutes.

reads by bytes per second appear to be very flat, with a few small fluctuations only a few times per day. The scale of the visualization when viewed with writes is confusing and suggests an inaccurate trend. Reads are not as variable as writes, but the STDEV is 18% of the mean value. Both reads and writes have high variability (Table 2). Initially writes appeared to have four data points that were outliers, as seen in the high spikes on the bottom (green) line in Figure 2. However, even with the four data points removed, the variability is high or 45% of the mean value (Table 2).

	Writes (with outliers removed)	**Writes (raw)**	**Reads**
Mean	18,799,954	22,726,801.95	544,754,417
STDEV	8,456,525	23,998,263.25	96,574,868
% of Mean	45%	106%	18%

Table 2: Variability in Read/Writes per bytes per second in the Ceph file system.

Even so, the high ingestion times don't always correspond with file system load. For example, the ingest time of 96 minutes comes just after a period of high write load. However, other high ingest times correspond to relatively low load. More study is needed to adequately attribute ingest times to file system load.

Data Characteristics

Another point of inquiry was to verify that the individual data files ingested were dissimilar enough to account for the processing variability. Each 2.5 million records file was analyzed for word and character counts. Because of an issue with one file processed, the result was a missing or incomplete value. This represented noise in the data, not an outlier (Aggarwal 2015). After normalizing the data, the underlying trend was more easily seen (see Appendix IV); the standard deviation in the tens and hundreds of millions looked large but was actually very small relative to the values' range: only 2% of the mean value. Looking at differences between words and characters did not reveal anything unexpected. The correlation between words and characters was very similar between files and fit a predictable relationship with an R^2 of .97 (Figure 3).

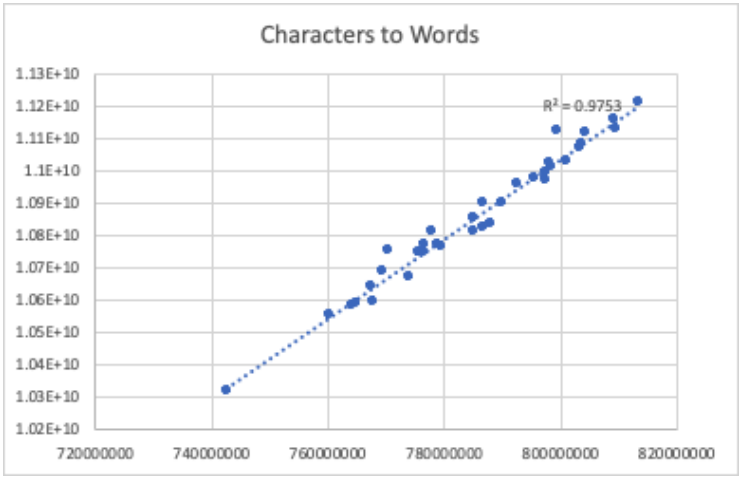

Figure 3: Characters per file were directly correlated to the number of words.

Data Pre-processing

With no obvious source for the ingestion time variability, attention was turned to preprocessing, *Hypothesis$_2$ - Ingestion time can be decreased through data preprocessing.* This hypothesis was investigated by the research computing developer on the team. As noted in the workflow description, duplicate tweets were found in many of the uncompressed .zip files. The developer had already tested pre-processing the tweet files to find and remove duplicates. However, at 2.5 million lines per file, just loading the file in any editor or running a command against it, was very time consuming. The anecdotal report was that the pre-processing took as much time as it saved on ingestion, rendering it a moot solution unless pre-processing was parallelized. This requires significant expertise and time investment to set up. Pre-processing experimentation was set aside in favor of other potential experiments.

Data Schema and Index Type

Though best practices on data insert was a favored hypothesis (***Hypothesis$_3$***), with the information that pre-processing or 'hitting' duplicate keys on ingestion was time intensive in this workflow, a new experiment was designed to avoid the duplicate tweet issue altogether. Rather than use the tweet ID as a unique key, an additional field could be added to act as the unique ID. Recent versions of PostgreSQL have a feature for a field to be an auto-incrementing integer. The database was recreated using the GNIP schema (see Appendix I), but the last three lines (60-62):

```
60. ALTER TABLE ONLY public.tweet ADD CONSTRAINT tweet_id_pk PRIMARY KEY (tweet_id);
61. CREATE UNIQUE INDEX tweet_id_uindex ON public.tweet USING btree (tweet_id);
62. CREATE INDEX user_id_uindex ON public.tweet USING btree (user_id);
```

were replaced by a line that creates a primary key that is an incrementing integer:

```
ALTER TABLE public.tweet ADD COLUMN unique_id SERIAL PRIMARY KEY
```

Hypothesis$_3$ and Revisiting Hypothesis$_1$

Upon dropping the constraint of a unique key based on the tweet ID, and replacing the primary key with a unique, auto-incrementing integer, there was a remarkable decrease in processing times (Table 3, Figure 4). One file only processed 1.4 million records and skewed results when included; this data was normalized by replacing the data noise with the mean value. Further investigation into what caused the ingest to fail before the file was complete showed a missing escape character to close a record. Data pre-processing includes searching for missing escape characters, but it was unsuccessful in this particular instance. The pre-processing script should be verified, as well as additional work on the ingestion script to allow for skipping lines with errors and finishing the file.

Lines/Records	**Records at Processing Start**	**Minutes to process**
2486936	0	15.62
2485910	2,486,936	14.87
2491324	4,972,846	15.76
2462280	7,464,170	15.23
2478868	9,926,450	16.36
2482803	12,405,318	16.57
1411266	*14,888,121*	*9.45*

Lines/Records	Records at Processing Start	Minutes to process
2486346	16,299,387	16.17
2488091	18,785,733	17.95
2486009	21,273,824	17.82
2485729	23,759,833	18.96
2490273	26,245,562	18.11

Table 3: Sample of processing times by number of database records and lines or records per file processed.

Even with the outlier time included, a new trend emerges (Figure 4). There is a slight trend towards higher processing times as the database size grows, with an R^2 of .175.

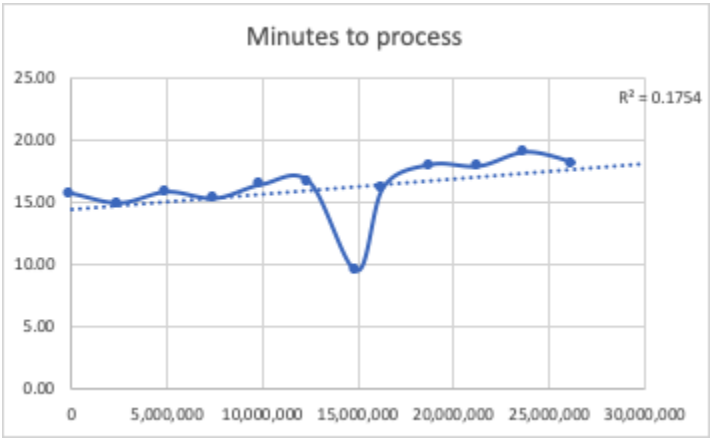

Figure 4: Processing times by records in the database prior to ingestion with the data noise included.

When the data is normalized and the mean value inserted for processing time (16.67 minutes) and number of lines (2,484,052 records) in its place, the correlation is even clearer (Table 4, Figure 5). The processing times show a correlation to records in the database, R^2 = .82.

Records at Processing Start	Minutes to process
0	15.62
2,486,936	14.87
4,972,846	15.76
7,464,170	15.23
9,926,450	16.36

Records at Processing Start	Minutes to process
12,405,318	16.57
14,888,121	16.67
17,372,173	16.17
19,858,519	17.95
22,346,610	17.82
24,832,619	18.96
27,318,348	18.11

Table 4: Normalized data, omitting noise

Figure 5: Times to process based on records at ingestion.

In the original workflow, to ingest 18.5 days of data it took 34.4 hours. With the simple change of primary key prior to data copy, the same period of data took 9.91 hours to copy into the database, or about 3.5 times faster. Examining sample speed increases between experiments, shows that the change in key value is responsible consistently for 230-330% increases (Table 5). The savings in cost and energy is significant, especially knowing that this sample is representative of activity that runs year-round on SDSC Cloud.

Database Size at Start	Minutes to process	w/o Tweet ID as Primary Key	% Time Improvement
36,197,823	67.43	23.04	293%
38,683,405	48.95	19.04	257%
41,169,195	56.62	18.88	300%

Database Size at Start	Minutes to process	w/o Tweet ID as Primary Key	% Time Improvement
43,655,140	52.79	22.78	232%
46,140,396	56.04	17.09	328%
48,626,048	55.44	18.00	308%

Table 5: Sample speed increases due to change in primary key.

Once the research workflow is adjusted based on database management administration best practices, $Hypothesis_1$ is proven true (Figure 5). However, the effect of number of records is very small compared to the overall processing times with a mean of 16.7 and a standard deviation of 1.3 minutes (8% of the mean time). This compares with the original run time mean of 56 minutes and a standard deviation of 7 minutes or 12% of the average run time.

Economic and Climate Impact from Compute Efficiency

In the United States alone, data centers consume nearly 2% of the total electricity load (Shehabi 2016). This computing load is estimated to require 73 billion kWh in 2020. A more aggressive estimate pegged US data center consumption at 90 billion kWk per year, which requires the equivalent of "requiring roughly 34 giant (500-megawatt) coal-powered plants" (Danilak 2017). With the global computing load estimated to be closer to 3% of the world's electricity supply or 416 terrawatts, with that consumption doubling every four years, it is critical efficiency factors larger into computing plans.

Real impact can be seen in the economic and energy costs saved through compute cycles avoided. In this work, simply by using a different primary key in the schema design, ingesting a year's worth of Twitter data in the workflow described would save 19.3 kW or the equivalent of CO_2 emissions from charging a smartphone 1,740 times or burning 15 pounds (6.8 kg) of coal (Table 8) (EPA 2020). For resource savings calculations, the following costs and energy consumption statistics were used (Table 6). Costs reflect SDSC external customer rates.

	Size	Cost per hour (USD), external rate	Watts (w) per hour
Heron Tweet collection	xe.large		
Tweet pre-processing	m1.medium	$0.116	
Social media database	R.1xlarge	$0.638	40w
Storage	1 TB volume	$0.00009135/GB/hour or $0.063/TB/hr	

Table 6: SDSC Cloud External Rates and Estimated Power Usage

Utility Costs and PUE

Power/utility costs are based on an average of $0.13/kW/hr. This is much cheaper than average power costs in California (USA) and remains low as the University of California San Diego generates 72% of its own electricity; the percentage fluctuates with demand, in summer demand is higher and the co-generation percentage is lower. Only requiring a portion of the campus energy needs from the local utility insulates the campus from rising and fluctuating costs (UCSD 2020). A more accurate way to account for power costs is to multiply the power consumed by the individual system by the energy rating of the data center. SDSC has a Power Usage Effectiveness (PUE) rating of 1.35. While an imperfect measurement that fails to account for hardware efficiency and other factors, it is the industry standard and current best practice for calculating the true cost of powering (computer) infrastructure in data centers (Horner 2016). This means for every unit of power consumed, it requires an additional 35% to maintain the environment, primarily the cooling and other infrastructure that provides a stable environment in a data center. For example, for computing equipment sitting in the SDSC data center, for every one kilowatt (kW) of energy consumed (per hour) at a rate of $0.13 (€0.12) for utility costs alone, accounting for the added load of cooling (PUE of 1.35), the actual cost to consume the 1 kw for one hour is ~$0.18 (€0.16).

Estimating Power Consumption in Shared Environments

Power consumption in a shared environment can be difficult to measure, but not impossible to estimate. The R.1xlarge cloud node that stores the database is one of ten identical nodes in a rack consuming 4kW on average (*4kW ÷ 10 nodes = 400w)*. Of the 400w attributable to the 1U node, the R.x1large instance uses 7% of the CPU and 12% of the RAM. The majority of energy consumption in the node is attributable to the CPU, but for estimation, one can attribute approximately 10% of the 1U node's power use to the R.x1large instance or 40w (*400w ÷ ~10% = 40w)*.

With these values, one can estimate the power savings attributable to the avoided compute cycles in the above experiment. The first savings estimate is based on the experiment run for 18.5 days of Twitter data (Table 7).

Sample 18.5 days of social media record ingestion

Primary Key	Time (hrs)	Compute Cost	Storage	Power (kW)	Power Cost	Total Cost	Savings	kW Saved
Tweet ID	34.4	$21.95	$2.17	1.4	$0.24	$24.36		
Integer	9.91	$6.32	$0.62	0.4	$0.07	$7.02	**$17.34**	**1.0**

Table 7: Economic and power savings based on 18.5 day sample used in the experiment.

This data was extrapolated to estimate the savings possible from ingesting one year of Twitter data (Table 8). A researcher paying for the cloud ingestion of Twitter data could save $342.11 in avoided compute costs and 19.3 kW of power usage.

Ingestion Costs for one year's worth of Twitter data

Primary Key	Time (hrs)	Compute Cost	Storage	Power (kW)	Power Cost	Total Cost	Savings	kW Saved
Tweet ID	678.7	$433.02	$42.76	27.1	$4.76	$480.54		
Integer	195.5	$124.74	$12.32	7.8	$1.37	$138.44	**$342.11**	**19.3**

Table 8: Economic and power savings estimated for ingestion using new unique key for one year of Twitter data.

Further Study

In the original workflow, the database schema separates users and tweets into two tables, whereas these experiments utilized a standard GNIP schema that kept both in one table. It is possible that the science drivers and characteristics of data, such as volume, will necessitate separating data into multiple tables. In these cases, random integers as keys may not perform as well for joins across tables. Further inquiry should be done to see if the findings here can improve the ingestion performance for user-defined schemas, especially ingestion across multiple tables.

Several areas of additional study should be investigated. 1. Additional experiments for query times using different index types should be tested. These might include indexes based on GIN (rather than B-tree), or custom extensions such as VODKA and RUM. These index types produce faster querying when combining complex column types, such as full-text phrase searches (Obe 2017). 2. Increase memory available on the RAM disk to test until ingestion cannot be improved, to attempt to find the true ingestion speed limitation attributable to the underlying, shared storage (Ceph file system) reads/writes. It may also be fruitful to assess use of RAM disks versus SSDs, especially savings in job time and cost. 3. Adapt this workflow to use with big data technologies such as Apache Hive over Hadoop to provide an SQL-like query interface while making use of other database architectures, especially those designed for big data needs. With the highest goal being increased ‘time to science’ for researchers, it will be important to factor learning curve needed to use the new methods into any decreased ingestion processing times.

Acknowledgements

This work would not have been possible without my advisor, Dr. Ines Dutra, and the support of SDSC’s Director, Dr. Mike Norman. Kevin Coakley and Colby Walsworth were instrumental in making cloud resources available, for providing access to underlying cloud logs, and suggestions on DBMS best practices. Kevin Coakley provided the workflow and scripts related to the Twitter workflow documented here and made improvements to the workflow that were suggested through our interviews and work together. Drs. Amarnath Gupta and Subhasis Dasgupta provided the use case for the workflow and were interviewed for this work.

Appendix

I. Schema.sql - Line numbers have been added for use in referencing portions of the database schema.

```
1. -- https://support.gnip.com/articles/relational-databases-part-4.html
2. --
3. -- https://developer.twitter.com/en/docs/tweets/data-
   dictionary/overview/tweet-object.html
4. --
5. CREATE TABLE public.tweet (
6.    tweet_created_at timestamp without time zone,
7.    tweet_id bigint NOT NULL,
8.    tweet_id_str character varying(80) NOT NULL,
9.    tweet_text text,
10.   tweet_source character varying(200),
11.   tweet_truncated boolean,
12.   tweet_in_reply_to_status_id bigint,
13.   tweet_in_reply_to_status_id_str character varying(80),
14.   tweet_in_reply_to_user_id bigint,
15.   tweet_in_reply_to_user_id_str character varying(80),
16.   tweet_in_reply_to_screen_name character varying(80),
17.   tweet_coordinates jsonb,
18.   tweet_place jsonb,
19.   tweet_quoted_status_id bigint,
20.   tweet_quoted_status_id_str character varying(80),
21.   tweet_is_quote_status boolean,
22.   tweet_quoted_status jsonb,
23.   tweet_retweeted_status jsonb,
24.   tweet_quote_count integer,
25.   tweet_reply_count integer,
26.   tweet_retweet_count integer,
27.   tweet_favorite_count integer,
28.   tweet_entities jsonb,
29.   tweet_extended_entities jsonb,
30.   tweet_favorited boolean,
31.   tweet_retweeted boolean,
32.   tweet_possibly_sensitive boolean,
33.   tweet_filter_level character varying(80),
34.   tweet_lang character varying(80),
35.   tweet_matching_rules jsonb,
36.   user_id bigint NOT NULL,
37.   user_id_str character varying(80) NOT NULL,
38.   user_name character varying(80),
39.   user_screen_name character varying(80),
40.   user_location character varying(200),
41.   user_derived jsonb,
```

```
42.    user_url character varying(200),
43.    user_description character varying(400),
44.    user_protected boolean,
45.    user_verified boolean,
46.    user_followers_count integer,
47.    user_friends_count integer,
48.    user_listed_count integer,
49.    user_favourites_count integer,
50.    user_statuses_count integer,
51.    user_created_at timestamp without time zone,
52.    user_profile_banner_url character varying(200),
53.    user_profile_image_url_https character varying(400),
54.    user_default_profile boolean,
55.    user_default_profile_image boolean,
56.    user_withheld_in_countries jsonb,
57.    user_withheld_scope character varying(80)
58. );
59. ALTER TABLE public.tweet OWNER TO postgres;
60. ALTER TABLE ONLY public.tweet ADD CONSTRAINT tweet_id_pk PRIMARY KEY
    (tweet_id);
61. CREATE UNIQUE INDEX tweet_id_uindex ON public.tweet USING btree (tweet_id);
62. CREATE INDEX user_id_uindex ON public.tweet USING btree (user_id);
```

Lines 60-62 were omitted and the following command (line A) was used instead to establish a unique, auto-incrementing key instead:

```
A.  ALTER TABLE public.tweet ADD COLUMN unique_id SERIAL PRIMARY KEY;
```

II. Once the data is pre-processed, it is copied into Postgres via this command:

> psql -h localhost -U postgres -W --command "\\copy public.tweet (tweet_created_at, tweet_id, tweet_id_str, tweet_text, tweet_source, tweet_truncated, tweet_in_reply_to_status_id, tweet_in_reply_to_status_id_str, tweet_in_reply_to_user_id, tweet_in_reply_to_user_id_str, tweet_in_reply_to_screen_name, tweet_coordinates, tweet_place, tweet_quoted_status_id, tweet_quoted_status_id_str, tweet_is_quote_status, tweet_quoted_status, tweet_retweeted_status, tweet_quote_count, tweet_reply_count, tweet_retweet_count, tweet_favorite_count, tweet_entities, tweet_extended_entities, tweet_favorited, tweet_retweeted, tweet_possibly_sensitive, tweet_filter_level, tweet_lang, tweet_matching_rules, user_id, user_id_str, user_name, user_screen_name, user_location, user_derived, user_url, user_description, user_protected, user_verified, user_followers_count, user_friends_count, user_listed_count, user_favourites_count, user_statuses_count, user_created_at, user_profile_banner_url, user_profile_image_url_https, user_default_profile, user_default_profile_image, user_withheld_in_countries, user_withheld_scope) FROM 'tweets_01.csv' DELIMITER ',' CSV QUOTE '|' ESCAPE '"';"

III. Processing times in minutes compared to number of records in the database prior to ingestion.

Database Size at Start	Minutes to process
36,197,823	67.43
38,683,405	48.95
41,169,195	56.62
43,655,140	52.79
46,140,396	56.04
48,626,048	55.44
51,112,609	50.36
53,599,799	43.12
56,086,810	58.45
58,573,610	64.49
61,059,010	57.41
63,544,592	44.34
66,030,382	47.58
68,516,327	54.60
71,001,583	63.44
73,487,235	51.57
75,973,796	54.05
78,460,807	62.51
80,947,607	59.46
83,433,007	64.67

IV. Tweet File Properties - the file value ‘5m-04-aa_tsu’ (noise) was replaced with the mean.

	Lines	Words	Characters	File path
	2486936	786809583	10824051372	5m-01-aa_tsu
	2485910	795689691	10975190898	5m-01-ab_tsu
	2491324	801012264	11025300350	5m-02-aa_tsu
	2462280	804336592	11114944355	5m-02-ab_tsu
	2478868	792772808	10955904268	5m-03-aa_tsu
	2482803	789963377	10897665169	5m-03-ab_tsu
	2486670	785250601	10852532776	5m-04-aa_tsu
	2486346	797446224	10992026011	5m-04-ab_tsu
	2488091	788165743	10834386551	5m-05-aa_tsu
	2486009	803569862	11080499028	5m-05-ab_tsu
	2485729	798638995	11012317981	5m-06-aa_tsu
	2490273	809126298	11159784409	5m-06-ab_tsu
	2484863	798189921	11022828424	5m-07-aa_tsu
	2490850	797495165	10966756250	5m-07-ab_tsu
	2491420	809478696	11125317048	5m-08-aa_tsu
	2490186	813494334	11211124791	5m-08-ab_tsu
	2490764	803342946	11069546867	5m-09-aa_tsu
	2492941	784955503	10813646660	5m-09-ab_tsu
	2492764	779567049	10764980642	5m-10-aa_tsu
	2491515	767722229	10591648439	5m-10-ab_tsu
	2488963	776620003	10744376991	5m-11-aa_tsu
	2488613	774062706	10668983243	5m-11-ab_tsu
	2486840	778997122	10768461647	5m-12-aa_tsu

	Lines	Words	Characters	File path
	2485790	799528915	11121589728	5m-12-ab_tsu
	2485894	786902641	10897089656	5m-13-aa_tsu
	2486275	776148175	10744877440	5m-13-ab_tsu
	2485582	778103119	10811653953	5m-14-aa_tsu
	2485790	775665438	10744723541	5m-14-ab_tsu
	2485945	764885657	10590187671	5m-15-aa_tsu
	2485256	767570466	10642572188	5m-15-ab_tsu
	2485652	764226177	10579847378	5m-16-aa_tsu
	2486561	776609695	10772911505	5m-16-ab_tsu
	2487190	770373893	10752641276	5m-17-aa_tsu
	2487011	769483994	10687158844	5m-17-ab_tsu
	2486800	760236270	10553747791	5m-18-aa_tsu
	2485400	742579488	10319904796	5m-18-ab_tsu
	2486670	785250601	10852532776	5m-19-aa_tsu
Mean	**2486670**	**785250601**	**10852532776**	
STDEV		**15807704.3**	**197550714**	
STDEV % from Mean		**2%**	**2%**	

Works Cited

Aggarwal, Charu C. Data mining: the textbook. Springer (2015).

Andrade, E., Nogueira, B. Performability Evaluation of a Cloud-Based Disaster Recovery Solution for IT Environments. *J Grid Computing* **17,** 603–621 (2019). https://doi.org/10.1007/s10723-018-9446-2

Barberá, Pablo, and Zachary C. Steinert-Threlkeld. "Social media as data generators." http://www.luigicurini.com/uploads/6/7/9/8/67985527/pb-zst-chapter.pdf Accessed 14 January 2020. 2019. Forthcoming in Curini, L., and Franzese, R. (eds) The SAGE Handbook of Research Methods in Political Science and International Relations, London: Sage (2019).

Beloglazov, Anton, Jemal Abawajy, Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. Future Generation Computer Systems, Volume 28, Issue 5, (2012). Pages 755-768, ISSN 0167-739X, https://doi.org/10.1016/j.future.2011.04.017.

Benbow, Nanette, Christine Kirkpatrick, Amarnath Gupta, Sean Young, and interested health officials. “A Mixed-Methods Iterative Process of Integrating and Developing Big Data Modeling and Visualization Tools for Public Health Officials.” SAGE Case Study (Submitted June 2019, under review).

Borodin, Andrey, et al. "Optimization of memory operations in generalized search trees of PostgreSQL." International Conference: Beyond Databases, Architectures and Structures. Springer, Cham (2017).

Ceph. https://ceph.io/ Accessed 15 January 2020.

CloudSim. Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne. http://www.cloudbus.org/cloudsim/ Accessed 2 February 2020.

Danilak, Radoslav. “Why Energy Is A Big And Rapidly Growing Problem For Data Centers.” Forbes. 15 December 2017. https://www.forbes.com/sites/forbestechcouncil/2017/12/15/why-energy-is-a-big-and-rapidly-growing-problem-for-data-centers/#22e754395a30 . Accessed 2 February 2020.

Environmental Protection Agency (EPA) Greenhouse Gas Equivalencies Calculator. https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator Accessed 11 January 2020.

Erradi, Abdelkarim, and Yaser Mansouri. "Online cost optimization algorithms for tiered cloud storage services." *Journal of Systems and Software* 160 (2020): 110457.

Filliez, Jeff. Interview regarding power consumption in the San Diego Supercomputer Center's data center. (2020).

Geschwinde, Ewald, and Hans-Jürgen Schönig. PostgreSQL developer's handbook. Sams Publishing (2002).

Graphite. https://graphiteapp.org/ Accessed 11 January 2020.

Grover, Raman, and Michael J. Carey. "Data Ingestion in AsterixDB." EDBT (2015).

Guo, Junan, Subhasis Dasgupta, and Amarnath Gupta. "Multi-Model Investigative Exploration of Social Media Data with boutique: A Case Study in Public Health." (Submitted on 24 May 2019). arXiv:1905.10482 [cs.DB]

Homer, Alex, John Sharp, Larry Brader, Masashi Narumoto, and Trent Swanson. Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications. Microsoft patterns & practices (2014).

Horner, Nathaniel, and Ines Azevedo. "Power usage effectiveness in data centers: overloaded and underachieving." The Electricity Journal 29.4 (2016): 61-69.

Neo4J. https://neo4j.com/. Accessed 14 January 2020.

Obe, Regina O., and Leo S. Hsu. PostgreSQL: Up and Running: a Practical Guide to the Advanced Open Source Database. " O'Reilly Media, Inc." (2017).

Oliveira, Danilo, et al. "Performability evaluation and optimization of workflow applications in cloud environments." *Journal of Grid Computing* 17.4 (2019): 749-770.

Oracle MySQL. https://www.mysql.com/ Accessed 15 January 2020.

PostgreSQL. The PostgreSQL Global Development Group, https://www.postgresql.org/. Accessed 15 January 2020.

SDSC Cloud. https://www.sdsc.edu/services/ci/cloud.html Accessed 11 January 2020.

Sefraoui, Omar, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing." International Journal of Computer Applications 55.3 (2012): 38-42.

Shehabi, Arman, et al. *United states data center energy usage report*. No. LBNL-1005775. Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), (2016).

Solr. Apache Solr. https://lucene.apache.org/solr/ Accessed 14 January 2020.
Steinert-Threlkeld, Zachary C. "Spontaneous collective action: Peripheral mobilization during the Arab Spring." American Political Science Review 111.2 (2017): 379-403.

Stonebraker, Michael, and Greg Kemnitz. "The POSTGRES next generation database management system." Communications of the ACM 34.10 (1991): 78-92.

Swift, OpenStack Object Storage. https://wiki.openstack.org/wiki/Swift Accessed 11 January 2020.

UCSD. “Clean Energy.” Sustainability. http://sustainability.ucsd.edu/focus/energy.html Accessed 11 January 2020.

Weil, Sage A., et al. "Ceph: A scalable, high-performance distributed file system." Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, (2006).

Zheng, Xiuwen, Qiyu Liu, and Amarnath Gupta. "Scalable Community Detection over Geo-Social Network." arXiv preprint arXiv:1906.05505 (2019).

Note: some text from “Related Work” was previously used in C. Kirkpatrick’s PhD Work Plan Application.