Github Mining for Golang: Topic Dynamics and Dependency Analysis

1st Justin Zhang
Computer Science and Software Engineering department
Auckland University of Technology
Auckland, New Zealand
schnell18@gmail.com

***Abstract*—The Go programming language and its ecosystem have been transforming the world of infrastructure software for container, block chain, cloud computing etc. Although there are extensive researches on open source projects by mining github repositories and the resulting public available datasets. There is no update-to-date golang specific dataset to answer questions such as what are golang's primary use cases, what are some popular libraries and how do they relate to each other, how do they evolve over times. In this paper, we fill the gap by building the ground truth about the third-party golang applications and libraries hosted on github by mining github to produce a dataset that captures the dependencies between the open source third-party scale golang projects. We explained how we solved challenges of collecting golang project data from github and how big data technology such as Hadoop MapReduce was utilized to aggregate open source golang project dependency data. The dataset as well as the programs to collect and process the data are published to facilitate future researches on golang. Based on the dataset, we conducted initial exploration on the evolution of the open source golang ecosystems on github and identified potential security threats introduced by third party library vulnerabilities. Then we propose some future research opportunities offered by the dataset.**

***Index Terms*—golang, Hadoop, MapReduce, dependency, github, Apache Parquet**

I. Introduction

Github is the forefront of open source software creation, dissemination and evolution. While golang is one of the most prominent open source system programming language which leverages Github to facilitate its engineering efforts, ranging from proposal management, defect tracking, version control, documentation, artifact build and so on. Apart from the core language runtime and standard library, a plethora of open source third-party libraries and applications are hosted on GitHub due to the fact that when the golang tool chain compiles golang program, it is able to resolve the libraries referenced by the program from source repositories like git [10]. This means github is an ideal place to host open source golang projects. Our research shows there are over 10,000 non-forking golang repositories with stars reaching 100. However, little is known about inter-dependencies of the vast amount of repositories and how do the themes of the golang applications evolve over last decade. In this research, our objective is to help the golang community make informed decision to choose appropriate third-party libraries by considering potential security threats and better understand the trend of golang applications. We'd like to share a practical data collection tool to study golang open source projects.

The primary research questions are as follows:

- RQ1: What are the primary use cases for golang and how do they evolve over time?
- RQ2: What are some popular third-party libraries?
- RQ3: How do security vulnerabilities affect the popular libraries?

To answer these research questions, we follow the process illustrated in Fig. 1. First, we identify golang repositories with at least 10 stars to form the base dataset. Then we collect *go.mod* file to build dependency dataset. The *go.mod* files are packed into a Hadoop-friendly data format called Apache Parquet [17], which are splittable columnar storage, so that we can use Hadoop MapReduce to calculate go module reference count, which enables us to

discover the top libraries.

Once these basic data are in place, we perform data aggregation to uncover golang application trends, top libraries and vulnerability analysis for top the 2000 libraries by comparing to the *Go Vulnerability Database* [21].

The data collection, cleanse, aggregation and visualization are automated by a combination of Python programs and jupyter notebooks. They are published on github for reference at [16].

The paper is organized as follows:

1) In section I, we give the background and research questions
2) In section II, we explain prior works on the mining software repository topic and what we can learn from these researches.
3) In section III, we discuss limitation of prior works and justify our research approach.
4) In section IV, we describe the data collection process in greater details and conduct fundamental data analysis on go application trends and top libraries discovery.
5) In section V, we make the data more digestible with visualization such as treemap and sunburst diagram.
6) In section VI, we explain how the golang module reference count is calculated by using big data technologies such as Hadoop MapReduce and Apache Parquet.
7) In section VII, we conclude this paper and suggest some future researches.

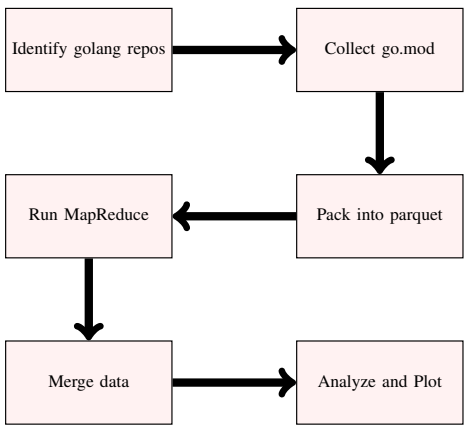

Fig. 1. Data process flow

II. Related Work

Mining Software Repository, referred to as MSR hereafter, is an empirical study of software engineering based on factual data collected from public coding collaboration platforms such as Github, Bitbucket, gitlab etc. There are large number of MSR researches published over last decade. The works can be roughly categorized into general-purpose dataset provision, data collection method and tool, domain specific study and critique of MSR researches.

In 2012, Gousios created GHTorrent [1], a pioneer effort to provide a general-purpose dataset containing history of github repository activities and offer ready-to-use MySQL and MongoDB query interfaces so that researchers can focus on their subject matter rather than being distracted by data collection. While GH Archive, studied by Markovtsev et al. [2], records the public GitHub timeline, providing raw event data in JSON format. Researchers can download the hourly-chunked JSON files for further analysis. The GH Archive project also offers the dataset via Google Big Query, which saves researchers lots of efforts to process the JSON data themselves.

Comparing to GH Archive, the GHTorrent's approach saves researchers efforts to access and process the data. The GH Archive's JSON data require further processing unless you use their Google BigQuery offering. On the other hand, GHTorrent requires substantial computing and storage capacity. Unfortunately, the official GHTorrent website is no longer in operation.

Dabic [6] et al. present GHS, which is a more recent effort to supply ready-to-use dataset to MSR researchers. The dataset is not as complete as GHTorrent or GH Archive, it covers 10 programming languages with 25 properties of each repository. The data is available in .csv format from a public accessible web interface with rich filtering criteria. The software behind the GHS dataset is open sourced. Researchers can customize the data collection tools and run their own instance for their unique requirements. Kaide et al. [3], created Argo, which uses GraphQL queries to fetch the communication histories from GitHub and plot line charts.

Comparing to GHTorrent and GH Archive, the GHS's scope is smaller, which is confined to a limited set of programming languages such C, C++, Java, golang etc. It only contains repository basic information and summary information such as number of stars, number of contributors. As a result, it is not sufficient for researches requiring access to detail data such as issues, commits, pull requests etc.

There are substantial field focused researches. He et al. [4] studied developer's acceptance of automated dependency update offered by the Github Dependabot. They got mixed findings and derived some key characteristics of an ideal dependency management bot. Montandon et al. [5] employed a combination of machine learning techniques to identify experts in software libraries with conclusion that the clustering method worked best for their test repositories.

Cosentino et al [11] conducted meta-analysis the MSR researches prior to 2016 and argue that some MSR researches are non-reproducible.

III. Opinions

The GHTorrent [1] is by far the most comprehensive dataset. The dataset is huge in size. However, it is based on MySQL data rather than big data platform such as Hadoop or Hive. As a result, the storage and computing capacity is limited and could hardly cope with ever-growing data on github. This may explain why the absence of data from recent years and the sudden decommissioning of the official website. While GH Archive [2] seems to have latest data, it, however, requires great efforts to sift its JSON data to get golang repository. And it definitely lacks the golang specific dependencies information.

As for the tools, although the Argo uses novel GraphQL API, the data it collects focus on communication history for known repositories. It is incapable of identifying list of repositories match certain criteria. And this tool is not publicly available. The GHS [6] provides to ready-to-use dataset, however, it doesn't include golang dependencies either.

The field specific researches have little coverage on golang. However, some research technique such as using machine learning to classify features is inspiring to conduct advanced analysis such as combine similar topic to reduce noise. Although there are dependency researches, for instance, Wagstrom et al. [13] presented a dependencies dataset for Ruby on Rail in 2013. The golang specific, update-to-date and ready-to-use dataset is still vacant.

Blincoe et al. [14] used cross-reference in comments to determine dependencies between two projects. Although the author did quantitative analysis and argued that this method is valid in 90% cases. However, this method is not sufficiently accurate as there is no validation on free-form cross-reference in comments which is subject to incomplete, incorrect reference or even omission. It can't reveal how the dependencies change as project evolves. In the case of golang, we have a solid foundation to build dependencies graph for golang modules as most golang projects declare dependencies precisely in the *go.mod* file which is strictly validated by golang compiler.

Through the review of related researches, we found the early stage of MSR studies usually involves identifying the software repositories to analyze in order to answer the research questions of interest. In this case, a general-purpose dataset can be of great aid. A general-purpose dataset has to capture static information as well as dynamic information such as commits, issues, pull requests, forks. It is huge dataset at terabyte scale as estimated by Gousios et al. [1] in 2012. The GHTorrent isn't based on big data platform which prevent it from scale to ever-growing size of github.

In this paper, we build the dataset from scratch in response to the gaps mentioned earlier. We need collect the *go.mod* files for each go repository and its published versions to build the inter-dependency relationship, which is unavailable from prior research or other public source. We would like to base our research on the latest data to reflect the constantly evolving and vibrant golang community, which is infeasible by using an existing dataset. The third drive to build the dataset on our own is that we want to come up a repeatable data collection and processing method which incorporates the Hadoop big data platform as we expect to extend future research beyond golang, which is supposed to be able to accommodate terabyte scale data and process data in parallel. Using pre-built dataset like GHTorrent,

GH Archive, GHS etc doesn't fulfill these goals.

IV. Data and Fundamental Statistical Analysis

The dataset we used is available at [24], which is one of the outcome of this research. In this section, we first describe the collection process in detail, then the initial statistical data analysis and library vulnerabilities analysis.

A. Data collection process

Although github offers convenient public REST and GraphQL API, collecting large scale dataset with long history of information such as repository, author, bot, issues, pull request, comment is still a non-trivial task. There are three major challenges to be solved in order to retrieve large search results from github. The first is 1000-limit issue [7], which discards records beyond 1000 in the result set of a particular query. The second is the rate-limit [8] issue, which prevents authenticated personal accounts from invoking API more than 5000 times per hour. When the client exceeds the rate limit, it is disconnected with HTTP status code 503. Without proper recover handling, data collection process is subject to frequent interruptions. And in the case of collection golang dependencies, the API invocation consists of published versions and the *go.mod* file retrieval, which is estimated to be around one million calls. Using one single personal account to collect the data takes a lot of time. The third is that the query results are paginated [9]. User has to issue multiple API calls to retrieve the complete query results over 100 records.

To work around the 1000-limit partial result issue, we come up a method to reduce the number of results below 1000 by dividing the repository creation time into consecutive time slots which is a series of 15-day windows, starting from 2008-1-1, ending with 2023-8-11.

To mitigate the rate-limit issue, we use the PyGithub [12] Python library which pauses API request to github when the number of requests exceeds the hourly limit. It waits until the next time slot when the allowed number of requests is reset and resumes the data collection process.

The precise criteria to identify the list of golang repositories are as follows:

1) main language is golang
2) created from 2008-1-1 to 2023-8-11
3) number of stars greater than or equal to 10

The second criterion is the key to successful retrieval of over 67923 repositories. The third criterion is necessary to reduce the noise of toy projects. The process to identify all golang repositories is automated by the Jupyter Notebook published on github [16].

Once the list of repositories are identified, we retrieve all *go.mod* files of the published versions for these repositories. Although the go module system has been around for over a decade, there are still a few legacy repositories which haven't migrated to the go module system. As a result, we excluded these repositories from dependencies analysis. Due to the 5000 per hour API call limit imposed on personal account by Github, We executed this part on an Oracle Cloud Infrastructure host and it took 10 days to complete, which downloaded 536,676 files matching the list of golang repositories we selected. In future research, This is definitely an area to improve by downloading in parallel using multiple accounts.

B. Initial Analysis

This golang repositories we collected from github were created between 2008-1-1 and 2023-08-11. Toy projects were excluded by applying the search to repository with at least 10 stars. This give us the total number of repositories as 67923.

To answer the research question 1, we first look at the basic statistics of number of repositories, stars, forks, watchers from perspective of topics. The data are presented in Table. I.

As shown in Table. I, the distribution of the repositories across topics are quite uneven and reveals an obvious long tail pattern. The 75% of topics include only 2 repositories, which uncovers the fact that the repository topics were extremely diverse. The stars, folks and watchers display similar pattern. There is no well-agreed rules to choose a proper topic. So some topic use similar words to represent the same underlying topic, for instance, *go* and *golang* are identical concept represented by different words.

The top 20 dominant topics are presented in Table. II, where the top topic *!!NOTOPIC!!* includes over half of the, repositories that we investigated.

TABLE I
BASIC STATISTICS OF GOLANG REPOSITORIES BY TOPIC

	repositories	stars	folks	watchers
count	34760	34760	34760	34760
mean	6	3111	423	3111
std	217	56073	8294	56073
min	1	10	0	10
25%	1	26	5	26
50%	1	93	18	93
75%	2	571	88	571
95%	13	9660	1144	9660
99%	55	46704	6040	46704
max	36120	6191678	880215	6191678

TABLE II
TOP 20 TOPICS STATISTICS OF GOLANG REPOSITORIES

topic	repositories	stars	folks	watchers
!!NOTOPIC!!	36120	4178023	880215	4178023
golang	14560	6079261	762961	6079261
go	10053	6191678	809695	6191678
kubernetes	2688	2091773	379772	2091773
cli	1578	770413	59230	770413
docker	1454	1296750	178332	1296750
hacktoberfest	1430	1531467	204678	1531467
security	720	605270	65562	605270
aws	673	272932	43763	272932
terraform	671	231869	58533	231869
prometheus	625	277799	52071	277799
golang-library	614	298596	31871	298596
api	613	205483	28452	205483
grpc	576	252929	34172	252929
linux	570	392975	41953	392975
blockchain	549	204671	79579	204671
http	494	368270	39614	368270
proxy	484	400128	60366	400128
redis	475	186015	25405	186015
database	465	512281	59053	512281

This indicates over half of repositories on github were not properly assigned a topic by their owners. Then follows the *go* and *golang* which were assigned to vast number of repositories.

To answer the research question 2, we identified the top 20 open source third-party golang libraries on github, which is presented in Table III. In this table, the first column *Repository* shows the name of repository on github. And the second column *Import Name* is the name which is actually referred to in the client code. It may differ from the github repository name when the author want a layer of indirection to decouple the library and source hosting choice so that they can switch the source code from github to other provider or on their own without breaking the client code using their library. According to Table III, the YAML libraries are exposed to client code using import name unrelated to github. The *Latest Version* and *Most Used Version* columns reveal that version lag in golang community is less common. Upgrading to latest versions in a timely manner is quite common practice. If we look at the *Refs* and *Stars* columns, we can discover that the actual use of the library and follow the library on github is not closely related. When library employs a import name other than github, the number of stars is significantly lower. Another interesting finding from Table III is that yaml libraries are widely used and the competition in the area is quite fierce as indicated by the presence of four players: *go-yaml/yaml*, *ghodss/yaml* and *kubernetes-sgis/yaml*.

The table is sorted by reference count, which is calculated by summarizing all *go.mod* files for all published versions of the 67923 repositories we selected in previous step. The calculation is implemented with Hadoop MapReduce and explained in Section VI.

C. Library Vulnerability Analysis

Similar to other modern programming language such as JavaScript, Python, Java etc, the golang community relies heavily on libraries to build applications. Vulnerabilities in the library, especially those being depended on indirectly, are prone to be ignored thus cause greater damages. To answer our research question 3, we use the golang dependency derived from the *go.mod* files collected in previous step and the *Go Vulnerability Database* [21] to identify potential vulnerabilities in libraries that are widely used. We select top 2000 golang libraries by reference count. Then we compare the fix version of the module in the *Go Vulnerability Database*, which is 441 at time of writing, to the latest version of the top 2000 golang libraries. If the libraries latest version is earlier than the fix version, we list the libraries as potentially vulnerable. Table.IV presents the top 20 libraries containing potential vulnerabilities.

As the Table.IV indicates, there are multiple libraries in kubernetes ecosystem containing vulnerabilities, for instance:

TABLE III
TOP 20 GOLANG LIBRARIES ON GITHUB AS OF SEPTEMBER 1, 2023

Repository	Import Name	Latest Version	Most Used Version	Refs†	Stars	Since‡
pkg/errors	github.com/pkg/errors	v0.9.1	v0.9.1	117518	8067	2015
spf13/pflag	github.com/spf13/pflag	v1.0.5	v1.0.5	76404	2158	2013
stretchr/testify	github.com/stretchr/testify	v1.8.4	v1.7.0	70935	20391	2012
go-yaml/yaml	gopkg.in/yaml.v2	v3.0.1	v2.4.0	57734	6413	2014
google/uuid	github.com/google/uuid	v1.3.1	v1.3.0	51492	4509	2016
ghodss/yaml	github.com/ghodss/yaml	v1.0.0	v1.0.0	32871	1001	2014
davecgh/go-spew	github.com/davecgh/go-spew	v1.1.1	v1.1.1	32671	5683	2013
sirupsen/logrus	github.com/sirupsen/logrus	v1.9.3	v1.8.1	31952	23118	2013
gorilla/mux	github.com/gorilla/mux	v1.8.0	v1.8.0	31801	19002	2012
mitchellh/go-homedir	github.com/mitchellh/go-homedir	v1.1.0	v1.1.0	30688	1330	2014
golang/protobuf	github.com/golang/protobuf	v1.5.3	v1.5.2	29396	9207	2014
go-yaml/yaml	gopkg.in/yaml.v3	v3.0.1	v3.0.1	27104	6413	2014
golang/mock	github.com/golang/mock	v1.7.0-rc.1	v1.6.0	24785	9060	2015
blang/semver	github.com/blang/semver	v4.0.0	v3.5.1+incompatible	22824	971	2014
gorilla/websocket	github.com/gorilla/websocket	v1.5.0	v1.4.2	22669	19655	2013
kubernetes-sigs/yaml	sigs.k8s.io/yaml	v1.3.0	v1.2.0	19190	192	2018
gogo/protobuf	github.com/gogo/protobuf	v1.3.2	v1.3.2	17967	5580	2014
dustin/go-humanize	github.com/dustin/go-humanize	v1.0.1	v1.0.0	17499	3714	2012
olekukonko/tablewriter	github.com/olekukonko/tablewriter	v0.0.5	v0.0.5	17293	3927	2014
fatih/color	github.com/fatih/color	v1.15.0	v1.13.0	16780	6478	2014

† Include references to all published versions and pseudo-versions
‡ Repository creation year

- *k8s.io/client-go*
- *k8s.io/apimachinery*
- *k8s.io/apiserver*
- *k8s.io/component-base*
- *k8s.io/kube-aggregator*

Further investigation should be carried out to validate the vulnerabilities as these libraries are widely used in mission-critical production environments.

V. AGGREGATION AND VISUALISATION

To answer the research question 1, we aggregate the golang repositories by the topics over years so that we can gain some insights on how the golang ecosystem evolves. We summarize over 67923 golang repositories by creation year and topics. We count the number of repositories in the each topic and year group and plot the dataset with the plotly [22] sunburst diagram as shown in Fig. 2. The plotting program is implemented in a jupyter notebook and can be found at [25]. We exclude topics which have repositories less than 60 to avoid clutters created by the vast number of less impactful repositories. There are some obvious patterns presented by Fig. 2. The first is the dominance of kubernetes, docker and cloud computing tools such

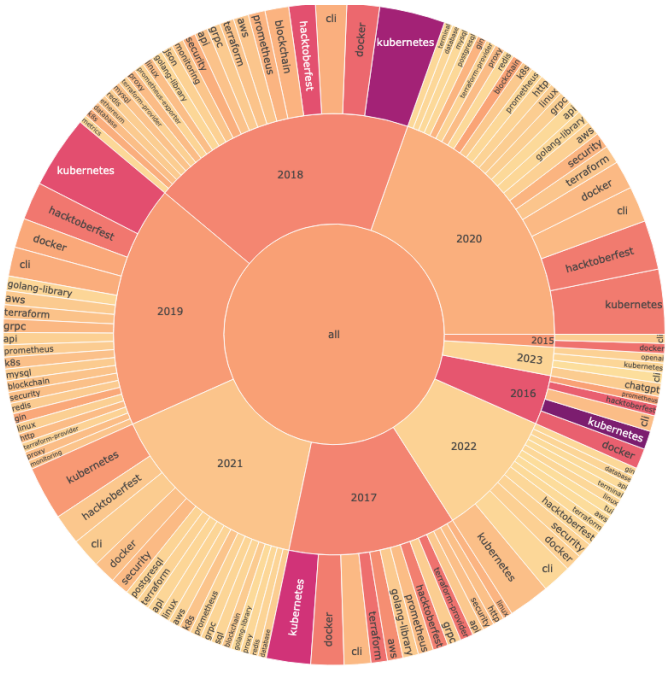

Fig. 2. Popular golang topic distribution over years on Github

TABLE IV
POTENTIAL VULNERABILITIES OF GOLANG LIBRARIES ON GITHUB AS OF SEPTEMBER 1, 2023

Library Name	Version	Vul IDs
github.com/golang/protobuf	v1.5.3	CVE-2023-24535
k8s.io/client-go	v12.0.0	CVE-2022-41723,CVE-2021-33194,CVE-2020-7919,CVE-2018-14632,CVE-2021-3121,GHSA-74fp-r6jw-h4mp,CVE-2019-11840,CVE-2022-27191,CVE-2020-29652,CVE-2020-9283,CVE-2023-3978,CVE-2019-11841,CVE-2021-43565,CVE-2022-41721,CVE-2022-41717,CVE-2022-27664,CVE-2019-9512,CVE-2021-44716,CVE-2021-31525
github.com/coreos/go-oidc	v3.6.0	CVE-2023-3978
github.com/onsi/gomega	v1.27.10	CVE-2023-3978
github.com/gin-gonic/gin	v1.9.1	CVE-2023-3978
github.com/jessevdk/go-flags	v1.5.0	CVE-2022-29526
github.com/pkg/browser	master	CVE-2022-29526
gopkg.in/src-d/go-git.v4	v4.13.1	CVE-2020-14040,CVE-2021-38561,CVE-2021-31525,CVE-2021-44716,CVE-2019-9512,CVE-2022-27664,CVE-2022-41717,CVE-2022-41721,CVE-2022-32149,CVE-2023-3978,CVE-2022-41723,CVE-2021-33194,CVE-2021-43565,CVE-2020-7919,CVE-2022-27191,CVE-2020-29652,CVE-2020-9283
google.golang.org/grpc	v1.59.0-dev	CVE-2023-3978
github.com/spf13/afero	v1.9.5	CVE-2022-32149
k8s.io/apimachinery	v0.29.0-alpha.0	CVE-2023-3978
github.com/vishvananda/netlink	v1.2.1-beta.2	CVE-2022-29526
k8s.io/apiserver	v0.29.0-alpha.0	CVE-2023-3978,CVE-2023-25151
go.opencensus.io	v0.24.0	CVE-2023-3978,CVE-2022-41717,CVE-2021-33194,CVE-2021-31525,CVE-2022-27664,CVE-2021-44716,CVE-2022-41721,CVE-2022-41723
k8s.io/component-base	v0.29.0-alpha.0	CVE-2023-25151
github.com/go-kit/kit	v0.13.0	CVE-2023-24535
github.com/google/gopacket	v1.1.19	CVE-2023-3978,CVE-2022-29526,CVE-2021-44716,CVE-2019-9512,CVE-2022-27664,CVE-2022-41717,CVE-2022-41721,CVE-2022-41723,CVE-2021-31525,CVE-2021-33194
k8s.io/kube-aggregator	v0.29.0-alpha.0	CVE-2023-3978
github.com/prometheus/common	v0.44.0	CVE-2023-3978
github.com/chzyer/readline	v1.5.1	CVE-2022-29526

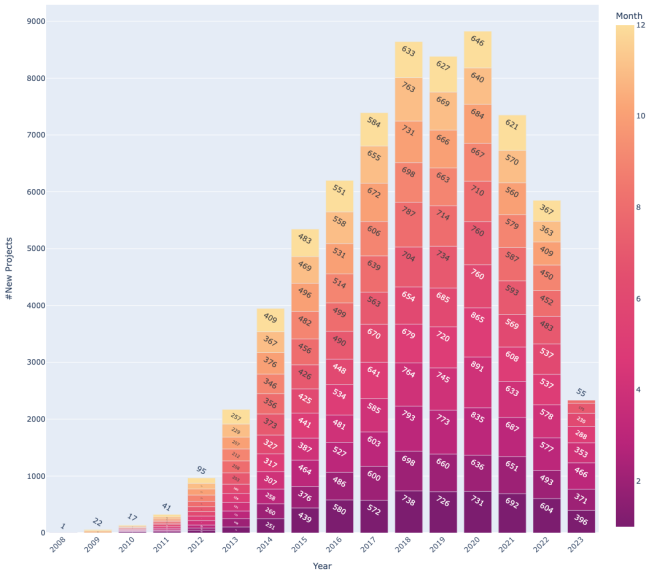

Fig. 3. Golang new repositories monthly increment since 2008

as AWS, terraform on the wheel in 2018, 2019, 2020 2021 etc. This reveals golang's primary use case is building infrastructure software for container orchestration and cloud computing. The second is new golang projects took advantage of open source community campaigns such as hacktoberfest [18] to encourage contribution and participation.

The topic evolution data are also illustrated as treemap presented in Fig. 4, which is capable of displaying more data points. Topics with over 15 repositories are included.

Judging by the diminishing areas of 2022 and 2023 comparing to the preceding years in the Fig.4 and the apparent downward trend indicated in Fig.3, we can safely conclude that new golang projects on github have declined considerably in 2022 and 2023. In the case of 2023, although there are still roughly 5 months to go, there is little chance to see dramatic rebounce in new projects in the coming months.

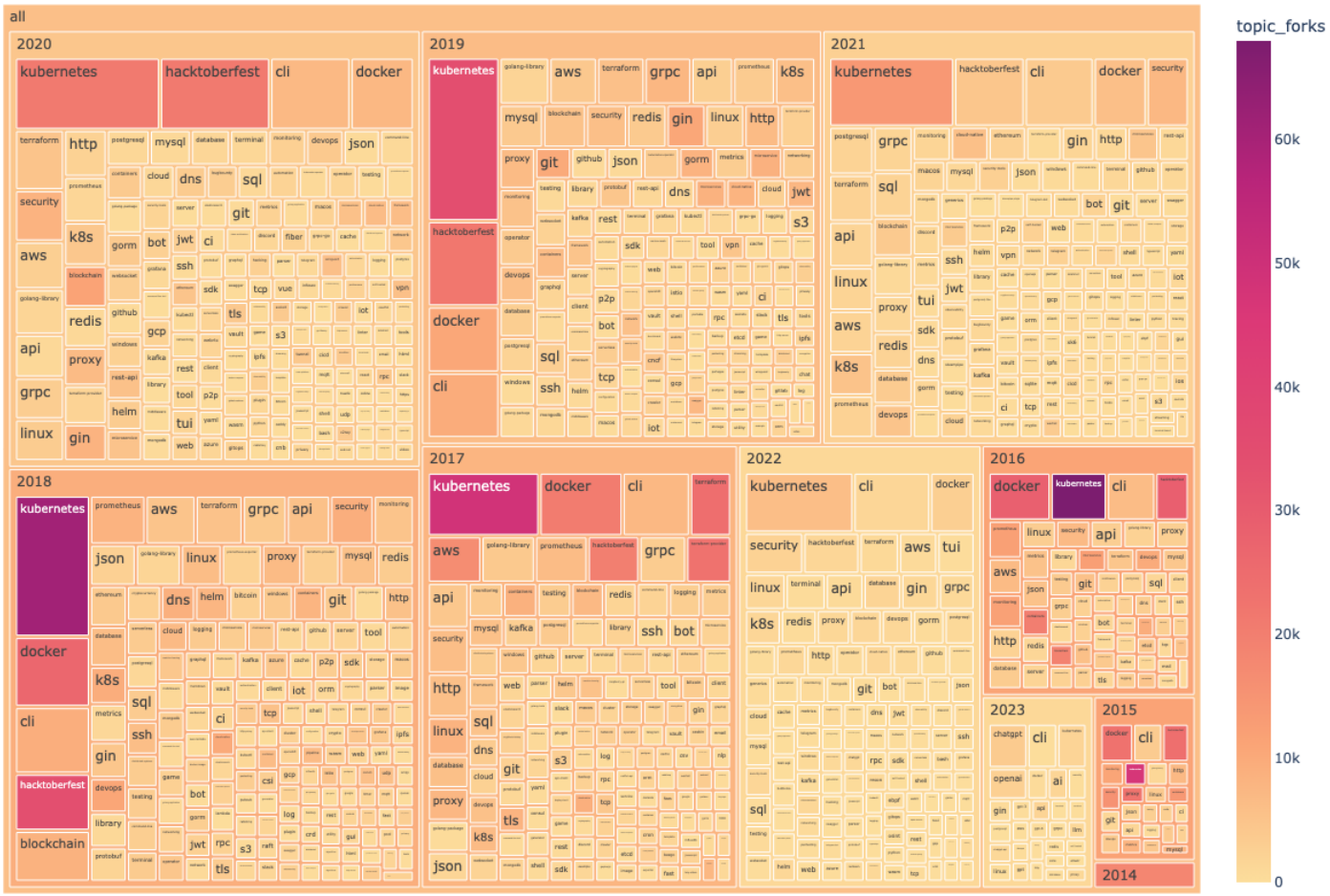

Fig. 4. Distribution over years for topic with repositories over 15

Another interesting dynamic is the ChatGPT and AI are dominant topics of 2023, indicating the focus of new golang projects has been shifting to AI.

VI. MapReduce Pseudo-code with Explanation

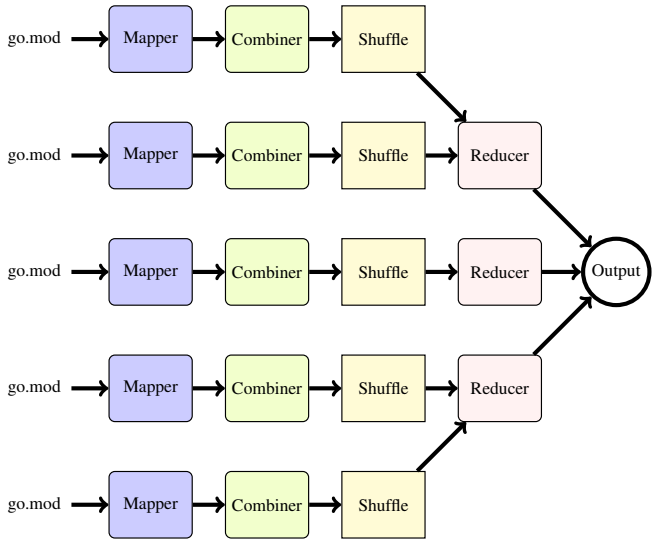

Fig. 5. MapReduce Architecture

In order to parse the dependencies information stored in over half million *go.mod* files and calculate the reference count of a particular library, we harness the big data tool Hadoop to speedup the processing. The overall architecture is depicted in Fig. 5. The rectangles with rounded corners represent the component we have to develop. The mapper is fed with the parquet record representing the *go.mod* and capable of receiving multiple input files from HDFS. The reducer performs the usual task to add up the reference count. The combiner is an optimization to accelerate the MapReduce job by taking advantage of the mathematical properties of the job as being communicative and associative. It shares the same logic as the reducer and reuses the code of reducer as will be presented later.

To explain how we parse the *go.mod* files, it's best to examine the relevant MapReduce code snippets. Before that, let's outline the primary steps of the whole process:

1) Encapsulate the *go.mod* files into multiple Apache Parquet files
2) Upload the parquet files into HDFS
3) Parse each *go.mod* file into pairs $\{(k_1, 1), (k_2, 1), \cdots, (k_n, 1)\}$ where k_i, $\forall i \in \{1, 2, \cdots, n\}$, is a string key representing the module being referenced
4) Use a combiner to sum the reference count from the same mapper
5) Sum the reference count in the reducer

The number of collected *go.mod* files is 536,676 with average size of 2670 bytes. We need to combine the small files into files in size close to 128M, which is default block size of Hadoop 3 HDFS [23], so that we can reduce excessive meta data that the name nodes have to maintain. Otherwise, it would slow down the data access, cause big storage waste and delay the completion of the MapReduce jobs. We pick the Apache Parquet data format, as various researches [19], [20] conclude parquet is efficient for big data processing in term of storage overhead and read speed. Our first attempt was packing all *go.mod* files into a single .parquet file. It generated a file around 250MB. Given the 128M block size of our Hadoop Cluster, it means we can only utilize two mappers in parallel. So we further split the *go.mod* files into 4 pieces, resulting 4 .parquet files to feed into the MapReduce jobs, which enables us to spin four mappers to count references in parallel at the cost of slight storage waste.

With that said, we present the pseudo-code for the mapper in listing 1, reducer in listing 2 and job configuration in listing 3 respectively. We explain these programs in greater details in the sections as follows.

A. Mapper

The mapper, named *GoDepMapper* as listed in the source Listing 1 is fed with a bunch of parquet records representing the belonging repository name, version, optional subpath and the content of *go.mod* file. In line 24, it extracts dependency module and version. As the focus of the mapper is to count direct library references, the indirect dependencies in the *go.mod* are ignored.

B. Reducer

The logic of the reducer is relatively simple. It adds up the reference counts for each key it receives and emits the total reference count for the module. In line 6 of Listing 2, we define the container of type *LongWritable* to store the aggregated reference count so that it can represent number as large as $2^{63} - 1$. With this arrangement, we can support even larger number of *go.mod* files.

The reducer program is also reused as the combiner to speedup MapReduce jobs.

C. Job Configuration

The source code of the job configuration class is listed in Listing 3. In line 15 – 22, we instruct the mapper to read input from multiple Apache Parquet files. As explain previously, we use parquet to compact the huge number of small *go.mod* files. If we put them into HDFS directly, it would stress the name nodes with excessive meta data management overhead and reduce the MapReduce performance.

In line 25, we optimize the dependency reference counting, being communicative and associative, by introducing the combiner to run along with the mapper on the same computing node in the Hadoop cluster so that we can relieve the burden of reducer and reduce data transfer overhead. The combiner has the same logic as the reducer, hence we reuse the *GoDepReducer*.

```java
package org.tinker.big.data.go.dep;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.parquet.example.data.Group;

import java.io.IOException;
import java.io.Serializable;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

public class GoDepMapper extends Mapper<Void, Group, Text, LongWritable> {

    private final Text outputKey = new Text();
    private final LongWritable outputValue = new LongWritable(1);

    @Override
    protected void map(Void key, Group value, Context context) throws IOException,
    ↪  InterruptedException {
        // Extract go.mod content from the Parquet record
        String goModContent = value.getString("content", 0);
        List<GoModuleRef> refs = parseGoModeFile(goModContent);

        for (GoModuleRef ref:refs) {
            outputKey.set(ref.moduleName);
            outputValue.set(1L);
            context.write(outputKey, outputValue);
        }
    }

    private List<GoModuleRef> parseGoModeFile(String goModContent) {
        // incomplete implementation
        String[] lines = goModContent.split("\n");
        return Arrays.stream(lines).filter(GoDepMapper::isDepDeclaration)
                .map(GoDepMapper::toGoModuleRef)
                .filter(Objects::nonNull)
                .collect(Collectors.toList());
    }

    private static boolean isDepDeclaration(String line) {
        return !line.contains("// indirect")
                && !line.contains("replace")
                && !line.contains("retract")
                ;
    }

    private static GoModuleRef toGoModuleRef(String line) {
        String[] comps = line.split("\\s+");
        if (comps.length == 2) {
            return new GoModuleRef(comps[0], comps[1]);
        }
        return null;
    }

    private static class GoModuleRef implements Serializable {
        public String moduleName;

        public String version;

        public GoModuleRef(String moduleName, String version) {
            this.moduleName = moduleName;
            this.version = version;
        }
    }
}

```

Listing 1: *Pseudo-code for mapper*

```java
package org.tinker.big.data.go.dep;

// imports omitted for brevity

public class GoDepReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    private final LongWritable outputValue = new LongWritable();

    @Override
    protected void reduce(Text key, Iterable<LongWritable> values, Context context) throws
    ↪  IOException, InterruptedException {
        int sum = 0;
        // Iterate over the values and calculate the sum
        for (LongWritable value : values) {
            sum += value.get();
        }
        outputValue.set(sum);
        context.write(key, outputValue);
    }
}

```

Listing 2: *Pseudo-code for reducer*

```java
package org.tinker.big.data.go.dep;

// imports omitted for brevity

public class GoDepCounter {

    public static void main(String[] args) throws Exception {
        if (args.length < 2) {
            System.err.println("You must specify output and at least one input");
            System.exit(-1);
        }
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJarByClass(GoDepCounter.class);
        for (int i = 1; i < args.length; i++) {
            MultipleInputs.addInputPath(
                job,
                new Path(args[i]),
                ParquetInputFormat.class,
                GoDepMapper.class
            );
        }
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(LongWritable.class);
        job.setCombinerClass(GoDepReducer.class);
        job.setReducerClass(GoDepReducer.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileOutputFormat.setOutputPath(job, new Path(args[0]));
        job.waitForCompletion(true);
    }
}

```

Listing 3: *Pseudo-code for Job Configuration*

VII. Conclusion and Future Issues

In this paper, we built a dataset of open source golang repositories and library dependencies to answer our research questions. We identified 67923 repositories created before 2023-8-11. We collected 536,676 *go.mod* files, from which the golang dependency dataset is derived.

We use topic changes to track golang ecosystem dynamics. Our findings are presented in Fig. 2 and Fig. 4 respectively. We conclude that the golang is primarily used to construct infrastructure softwares to support container orchestration, as demonstrated by kubernetes, web applications, RPC, blockchain and cloud computing tool such as awscli, Terraform. There is shift of focus from infrastructure to AI in new projects created in 2023.

The top 20 popular golang third-party libraries are presented in Table. III. The security threats of these libraries are identified and the top 20 potential affected libraries are presented in Table IV for further study. The vulnerability analysis may has false positive as it just matches the version without further consider the actual reference to affected functions. For future study we should conduct deeper validation of existence of vulnerabilities by checking the source code with vulnerability scan tools.

For further researches, we propose to conduct new explorations as follows:

1) Improve the data collection tool to grab data from github in parallel by using multiple accounts
2) Employ text classify technique based on text similarity to combine similar topics to reduce noise
3) Model the dependency using graph and analyze most influential nodes to evaluate security vulnerability blast radius and module couplings

References

[1] G. Gousios and D. Spinellis, 'GHTorrent: Github's data from a firehose', in 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), Jun. 2012, pp. 12–21. doi: 10.1109/MSR.2012.6224294.

[2] V. Markovtsev and W. Long, 'Public Git Archive: A Big Code Dataset for All', in 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), May 2018, pp. 34–37.

[3] K. Kaide and H. Tamada, 'Argo: Projects' Time-Series Data Fetching and Visualizing Tool for GitHub', in 2022 23rd ACIS International Summer Virtual Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Summer), Jul. 2022, pp. 141–147. doi: 10.1109/SNPD-Summer57817.2022.00032.

[4] R. He, H. He, Y. Zhang, and M. Zhou, 'Automating Dependency Updates in Practice: An Exploratory Study on GitHub Dependabot', IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 4004–4022, Aug. 2023, doi: 10.1109/TSE.2023.3278129.

[5] J. E. Montandon, L. Lourdes Silva, and M. T. Valente, 'Identifying Experts in Software Libraries and Frameworks Among GitHub Users', in 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), May 2019, pp. 276–287. doi: 10.1109/MSR.2019.00054.

[6] O. Dabic, E. Aghajani, and G. Bavota, 'Sampling Projects in GitHub for MSR Studies', in 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), May 2021, pp. 560–564. doi: 10.1109/MSR52588.2021.00074.

[7] About Search (2023) Github. Available at: https://docs.github.com/en/free-pro-team@latest/rest/search/search?apiVersion=2022-11-28#about-search (Accessed: 20 August 2023).

[8] Rate Limit (2023) Github. Available at: https://docs.github.com/en/rest/overview/resources-in-the-rest-api?apiVersion=2022-11-28#rate-limits (Accessed: 20 August 2023).

[9] About pagination (2023) Github. Available at: https://docs.github.com/en/rest/guides/using-pagination-in-the-rest-api?apiVersion=2022-11-28#about-pagination (Accessed: 20 August 2023).

[10] Go Modules Reference (2023) go.dev. Available at: https://go.dev/ref/mod (Accessed: 20 August 2023).

[11] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, 'Findings from GitHub: Methods, Datasets and Limitations', in 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), May 2016, pp. 137–141.

[12] Jacques, V. (2023) PyGithub, PyPI. Available at: https://pypi.org/project/PyGithub/2.0.0rc1/ (Accessed: 23 August 2023).

[13] P. Wagstrom, C. Jergensen, and A. Sarma, 'A network of Rails a graph dataset of Ruby on Rails and associated projects', in 2013 10th Working Conference on Mining Software Repositories (MSR), May 2013, pp. 229–232. doi: 10.1109/MSR.2013.6624033.

[14] K. Blincoe, F. Harrison, and D. Damian, 'Ecosystems in GitHub and a Method for Ecosystem Identification Using Reference Coupling', in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, May 2015, pp. 202–211. doi: 10.1109/MSR.2015.26.

[15] Go Module Mirror, Index, and Checksum Database(2023) golang.org. Available at: https://index.golang.org/ (Accessed: 25 August 2023).

[16] Jupyter notebook for golang github mining. (2023) github. Accessed: Aug. 27, 2023. [Online]. Available: https://github.com/schnell18/play-data-science/blob/master/msr-golang/identify-golang-repos.ipynb

[17] Apache Parquet Website (2023) apache.org. Available at: https://parquet.apache.org/ (Accessed: 20 August 2023).

[18] Hacktoberfest Participation (2023) Hacktoberfest. Available at: https://hacktoberfest.com/participation/ (Accessed: 20 August 2023).

[19] V. Belov, A. Tatarintsev, and E. Nikulchev, ‘Choosing a Data Storage Format in the Apache Hadoop System Based on Experimental Evaluation Using Apache Spark’, Symmetry, vol. 13, no. 2, Art. no. 2, Feb. 2021, doi: 10.3390/sym13020195.

[20] A. Boufea, R. Finkers, M. van Kaauwen, M. Kramer, and I. N. Athanasiadis, ‘Managing Variant Calling Files the Big Data Way: Using HDFS and Apache Parquet’, in Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, in BDCAT ’17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 219–226. doi: 10.1145/3148055.3148060.

[21] Go Vulnerability Database (2023) go.dev. Available at: https://pkg.go.dev/vuln/ (Accessed: 20 August 2023).

[22] Plotly Express (2023) plotly. Available at: https://plotly.com/python/plotly-express/ (Accessed: 20 August 2023).

[23] Apache Hadoop Express (2023) apache. Available at: https://hadoop.apache.org/docs/r3.1.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (Accessed: 20 August 2023).

[24] Golang github repositories and go.mod of all vers (2023) kaggle. Available at: https://www.kaggle.com/datasets/schnell18/golang-github-repositories-and-gomod-of-all-vers/code?datasetId=3688822 (Accessed: 1 September 2023).

[25] Github golang topics evolution plots (2023) kaggle. Available at: https://www.kaggle.com/code/schnell18/github-golang-topics-evolution-plots (Accessed: 1 September 2023).