DRAT: An Unobtrusive, Scalable Approach to Large Scale Software License Analysis

Chris A. Mattmann[1,2], Ji-Hyun Oh[1,2], Tyler Palsulich[1*], Lewis John McGibbney[1], Yolanda Gil[2,3], Varun Ratnakar[3]

[1]Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109 USA
mattmann@jpl.nasa.gov

[2]Computer Science Department
University of Southern California
Los Angeles, CA 90089 USA
{mattmann,jihyuno}@usc.edu

[3]USC Information Sciences Institute
University of Southern California
Marina Del Rey, CA
{gil,varunr}@ isi.edu

Abstract— **The Apache Release Audit Tool (RAT) performs software open source license auditing and checking, however RAT fails to successfully audit today's large code bases. Being a natural language processing (NLP) tool and a crawler, RAT marches through a code base, but uses rudimentary black lists and white lists to navigate source code repositories, and often does a poor job of identifying source code versus binary files. In addition RAT produces no incremental output and thus on code bases that themselves are "Big Data", RAT could run for e.g., a month and still not provide any status report. We introduce Distributed "RAT" or the Distributed Release Audit Tool (DRAT). DRAT overcomes RAT's limitations by leveraging: (1) Apache Tika to automatically detect and classify files in source code repositories and determine what is a binary file; what is source code; what are notes that need skipping, etc. (2) Apache Solr to interactively perform analytics on a code repository and to extract metadata using Apache Tika; and finally (3) Apache OODT to run RAT on per-MIME type (e.g., C/C++, Java, Javascript, etc.) and per configurable K-file sized chunks in a MapReduce workflow. Each Mapper task is an instance of RAT running on a K-file sized per Multipurpose Internet Mail Extensions (MIME) type chunk (split using Tika) and each mapper produces and incremental and intermediate log file; and where the Reducer aggregates the individual log files.**

Keywords— DRAT, Open source, Software license auditing

**** Palsulich performed this work while at NASA JPL; currently at Google***

I. Introduction

Due to an exponential increase in the volume and complexity of data, software is undergoing rapid development to keep pace and to effectively manage and analyze the data at today's scale. Above all, we are facing an increasing number of open source projects hosted in public repositories across many domains. As such the license of open source software becomes critical to grant everyone legally appropriate permission to freely use, modify, and distribute the open source software [1]. There exist more than 60 licenses, such as Berkeley Software Distribution (BSD), General Public License (GPL), MIT license, the Apache License, version 2 ("ALv2"), and so on, approved by Open Source Initiative (OSI) for complying with open source definition, however, there exist slight differences among these licenses [2]. For instance, GPL is a "copyleft" license that only allows derivative works under the original license, whereas MIT license is a "permissive" license that grants the right to sublicense the code under any kind of license [2]. This difference could affect architectural design of the software. Furthermore, circumstances are more complicated when people publish software under the multiple licenses. Therefore, an automated tool for verifying software licenses in code bases is highly desired.

As a part of the Apache Software Foundation (ASF) project, Apache Creadur [3], a Release Audit Tool (RAT) was developed especially in response to demand from the Apache Incubator and its dozens of projects. All codes donated from individuals or organizations to ASF should go through the Apache Incubator as an entry path. The Apache Incubator ensures that the software has correct open source licenses [4]. Therefore, the primary function of the RAT is automated code auditing and open-source license analysis focusing on headers. RAT is a natural language processing tool written in Java to easily run on any platform and to audit code from many source languages (e.g., C, C++, Java, Python, etc.). RAT can also be used to add license headers to codes that are not licensed [5].

In the summer of 2013, our team ran Apache RAT on source code produced from the Defense Advanced Research Projects Agency (DARPA) XDATA national initiative whose inception coincided with the 2012 U.S. Presidential Initiative in *Big Data*. XDATA brought together 24 performers across academia, private industry and the government to construct analytics, visualizations, and open source software mash-ups that were transitioned into government projects and to the defense sector. XDATA produced a large Git repository consisting of ~50,000 files and 10s of millions of lines of code. DARPA XDATA was launched to build a useful infrastructure for many government agencies and ultimately is an effort to avoid the traditional government-contractor software pipeline in which additional contracts are required to reuse and to unlock software previously funded by the government in other programs.

All XDATA software is open source and is ingested into DARPA's Open Catalog [6] that points to outputs of the program including its source code and metrics on the repository. Because of this, one of core products of XDATA is the internal Git repository. Since XDATA brought together open source software across multiple performers, having an understanding of the licenses that the source codes used, and their compatibilities and differences was extremely important and since there repository was so large, our strategy was to develop an automated process using Apache RAT.

We ran RAT on 24-core, 48 GB RAM Linux machine at the National Aeronautics and Space Administration (NASA)'s Jet Propulsion Laboratory (JPL) to produce a license evaluation of the XDATA Git repository and to provide recommendations on how the open source software products can be combined to adhere to the XDATA open source policy encouraging permissive licenses. Against our expectations, however, RAT failed to successfully and quickly audit XDATA's large Git repository. Moreover, RAT provided no incremental output, resulting in solely a final report when a task was completed. RAT's crawler did not automatically discern between binary file types and another file types. It seemed that RAT performed better by collecting similar sets of files together (e.g., all Javascript, all C++, all Java) and then running RAT jobs individually based on file types on smaller increments of files (e.g., 100 Java files at a time, etc).

The lessons learned navigating these issues have motivated to create "DRAT", which stands for "Distributed Release Audit Tool". DRAT directly overcomes RAT's limitations and brings code auditing and open source license analysis into the realm of *Big Data* using scalable open source Apache technologies. DRAT is already being applied and transitioned into the government agencies. DRAT currently exists at Github under the ALv2 [12]. In this paper, we will describe the DRAT in detail. The remainder of this paper is structured as follows. Section 2 documents DRAT's architecture and its workflow. Section 3 presents the results of running DRAT on several national repositories and on its performance. We present related work in section 4 and draw conclusions and provide future work in final section.

II. DRAT ARCHITECTURE

DRAT is a MapReduce (M/R) style [7] RAT workflow that runs on top of Apache Object Oriented Data Technology (OODT) [8], a scientific data processing, acquisition, and dissemination system. M/R, a methodology for processing a large amount of unstructured data with a parallel, distributed algorithm, consists of two components: "mapper" and "reducer". The map procedure takes an input files and generates a set of *intermediate* outputs. The reduce procedure, in turn, consolidates all the intermediate results from the mapper into one final result. In DRAT, the "mapper" is RAT, and the "reducer" is a log collector to combine all the intermediate outputs into a global RAT report that can be used for stats generation. DRAT leverages Apache Tika [9] as a splitter to automatically discern file Multipurpose Internet Mail Extensions (MIME) types. As one of top-level projects in the Apache Software Foundation (ASF), Tika is a tool for detecting MIME type and extracting metadata from a wide range of file types based on Internet Assigned Numbers Authority (IANA) registry. Fig. 1, for instance, shows that the automated refinement of files in the XDATA Git repository using Tika (lower in Fig 1) is comparable to the classification resulted from using *grep* and *find* commands (upper in Fig. 1).

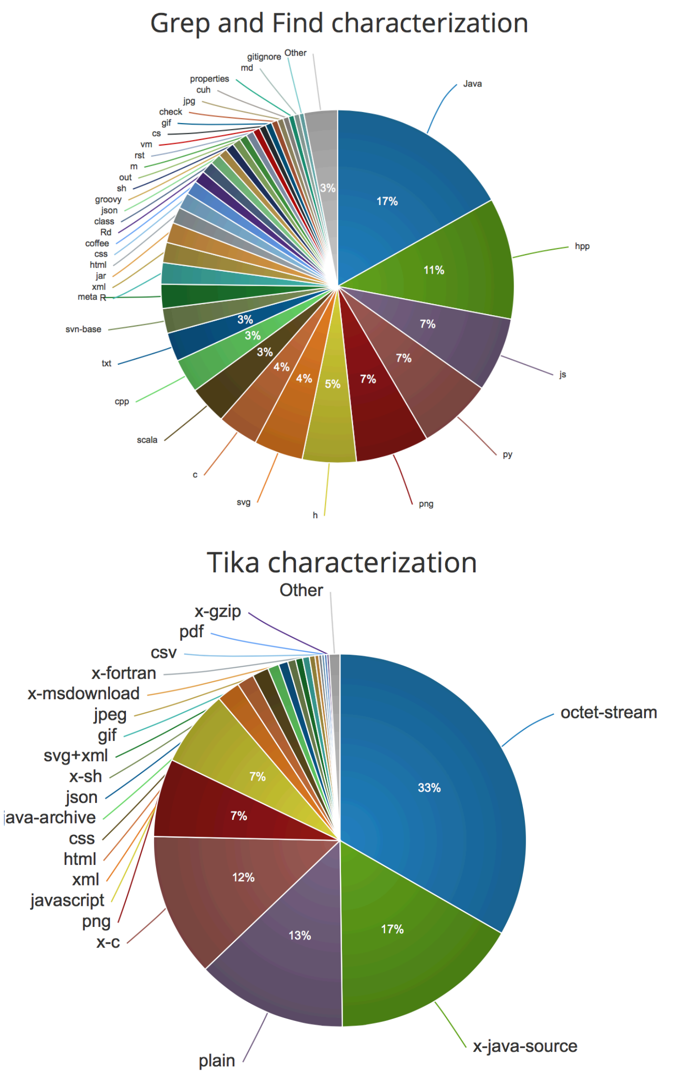


Figure 1. Distribution of formats of files from XDATA Git repository identified by grep and find command (upper) and by Tika (lower).

Although newer versions of Tika add support for new document formats, it is impossible to support all the file formats in the world. Thus, Tika is extensible allowing users to customize the tika-mimetypes.xml configuration file to add new MIME types. That said, Tika currently supports 1400+ MIME types and is one of the most comprehensive MIME repositories that exist.

Figure 2 depicts a schematic representation of the DRAT workflow. A source repository is ingested into Apache OODT

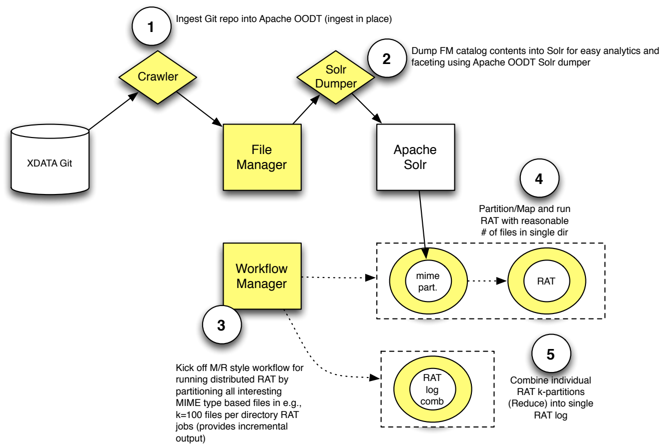

1
Ingest Git repo into Apache OODT (ingest in place)
Crawler
XDATA Git
File Manager
2
Solr Dumper
Dump FM catalog contents into Solr for easy analytics and faceting using Apache OODT Solr dumper
Apache Solr
4
Partition/Map and run RAT with reasonable # of files in single dir
Workflow Manager
mime part.
RAT
3
Kick off M/R style workflow for running distributed RAT by partitioning all interesting MIME type based files in e.g., k=100 files per directory RAT jobs (provides incremental output)
RAT log comb
5
Combine individual RAT k-partitions (Reduce) into single RAT log

Figure 2. Automated software metadata and license checking extraction architecture

as a start, and files and codes in the repository are catalogued in place. The MIME type of file is classified using Tika and in addition file metadata is extracted (ingestion date/time, source location, destination location, filename, etc.). After initial cataloging, the Tika file and code metadata is dumped into the Apache Solr search engine, to make it available for analytics and characterization in a distribution fashion. Solr is a scalable, distributed search server built on top of Apache Lucene, and our DRAT workflow reads metadata and file information directly from Solr. Once the code information is available in Solr the M/R style workflow gets started using Apache OODT: the mapper's job runs RAT on a 100 files (configurable) of the same MIME type per directory partitioned by Tika. The incremental output from the map procedure is aggregated by using RAT log combiner and OODT ensures all mappers and reducers are scheduled.

III. EXPERIMENTS AND EVALUATION

DRAT was run across the entire XNET Git repository. The result of license auditing is presented in Fig. 3. It displays the counts of Notes, Binaries (no license), Archive (tar/zip, no license), Standards (non-Apache, OSI approved license e.g., BSD, MIT, GPL, etc), Apache (ALv2 licensed), Generated (either source or binary files), and Unknown (Non-discernible license) files in the repository identified by DRAT – note this is adapted from RAT's existing output log format. Of the 19,491 source code files in the repository, 53% (10,271) use OSI approved licenses; 12% (2,398) have ALv2; 35% (6,795) have unknown licenses that requires further analysis. Of the initial 50,000 flies, only about half were actual source code; the remaining files were binary, and ignored by DRAT.

DRAT overcomes the limitations of RAT by 1) producing incremental output, 2) using Apache Tika for automated black and white lists by file type/MIME type, 3) using Apache Solr to analyze code repositories and search for files, and 4) using Apache OODT and M/R to scale out on large code bases.

One of the most remarkable improvements with DRAT is a dramatic reduction of total run time. As an example, Fig. 4 shows results of 3 experiments we carried out on the basis of DARPA XDATA Git repository in which we measured each of DRAT's stages: (1) ingestion in place (*Ingest*); (2) Solr dumping (*Solr Dumper*); (3) MIME partitioning with Tika (*MIME partitioner*); and (4) running RAT as a Mapper (*RAT*) and in which we computed the average time of the three runs. Note that we do not provide details on the reduce step since it represents a small contribution to the total time (~1 second). The horizontal axis shows major steps that consist of DRAT and the vertical axis shows execution time for each step. The average DRAT total run time is about 2 hours on a Macbook with dual quad cores, and 8 GB RAM, while a single job ran in 4 weeks on a fairly large machine based on more than 24 cores with 2GB of RAM per core as mentioned previously.

IV. RELATED WORK

This work applies to NASA and DARPA code auditing activities and more generally to software development as a whole in any agency, Federally Funded Research and Development Center (FFRDC), private industry, and so on. Furthermore, an official memorandum released by the Office of Science and Technology Policy (OSTP) in February 2013 encourages publishing results of federally funded research for free availability to the public. Scientific software arising from research is also an important asset of research results, which should be freely accessed and broadly disseminated.

Utilizing DRAT, we conducted an exploratory study aiming to develop a framework for software stewardship in geosciences to empower scientists to manage their software as valuable scientific assets in an open and transparent way. Geoscience, including geology, hydrology, meteorology, oceanography and so on, is a field with high reliance on computing to simulate real-world physical rules and principles. We used DRAT to perform license verification of scientific software deposited in Computational Infrastructure for Geodynamics (CIG) repository (http://geodynamics.org/cig/software/). CIG is a community driven organization focusing on developing and distributing software for geophysics and related fields. Codes donated by geoscientists in CIG are published under open source licensing. Its repository hosts more than 500 thousand open source files with 10s of millions of lines of code in a wide range of disciplines in geodynamics and computational

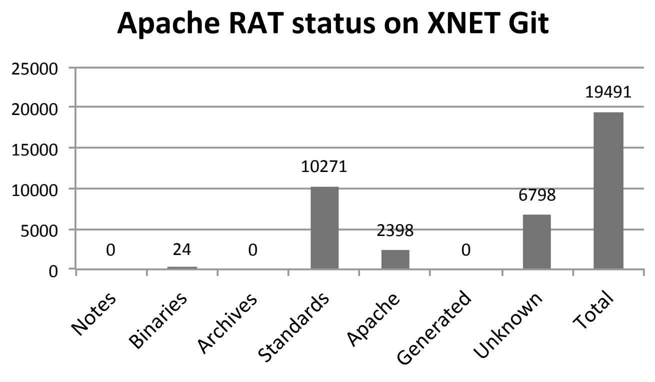


Figure 3. The number of files categorized by license type in DARPA XDATA Git repository. Standards are OSI approved licenses and Apache indicates ALv2 licenses.

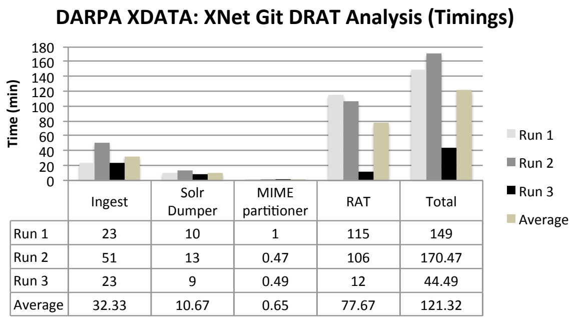
Table 1. The number of files cateogrized by license type in CIG repository

Notes	Binaries	Archives	Standards	Apache	Generated	Unknown	Unanalyzed
0 (0%)	38 (0%)	0 (0%)	135,230 (23%)	1 (0%)	0 (0%)	134,949 (23%)	318,862 (54%)

science. It took approximately 33 hours (1,980 minutes) to run DRAT on CIG repository as shown in Fig. 5. This is nearly 16 times as long as it took to run DRAT on the entire XNET Git composed of nearly 50,000 files and 10s of millions of lines of code for DARPA XDATA. Table 1 represents the result of DRAT analysis on CIG and Fig. 5 shows running time.

Over 54% of the CIG code base as put up on the CIG site are not code files, therefore, they are not analysed. While 23% of the files are unlicensed source code, 23% of the files are licensed using standards, i.e., BSD, MIT, OSI approved licenses. Thus, roughly half of the actual source code for CIG is either unlicensed or has unrecognizable license.

We also executed DRAT on Penn State Integrated Hydrologic Model (PIHM) package that consists of a few source codes. This run takes only 10 minutes. Comparing runtime of DRAT based on the different size of code bases,



	Ingest	Solr Dumper	MIME partitioner	RAT	Total
Run 1	23	10	1	115	149
Run 2	51	13	0.47	106	170.47
Run 3	23	9	0.49	12	44.49
Average	32.33	10.67	0.65	77.67	121.32

Figure 4. Execution time for M/R style distributed RAT runs to completion

DRAT scales linearly in the size of the code base.

In trying to expand the applicability of DRAT to broader geoscience fields, DRAT has been run on software used in Paleoclimatology community. National Oceanic and Atmospheric Administration (NOAA) National Climatic Data Center (NCDC) provides a catalogue of about 300 widely used software tools for Paleoclimatology [10] interoperating with existing software repositories in Paleoscience domain. The catalogue contains not only complex model source codes developed by large institutes but relatively small pieces of source code generated by individual scientists for ancillary tasks related to their studies.

We have focused on the latter in particular because the scientist-generated software tends to be less licensed than the software created by large institutes due to the lack of awareness of software licensing among scientists [11]. As such, 27 software packages coded in diverse languages, such as Java, R, Fortran, and Matlab, were selected from the Paleoclimatology software catalog for running through DRAT.

The performance of DRAT differs from our other experiences reported in the paper. In particular, DRAT fails to run properly for the software packages that consists of source codes written in Matlab, widely used in Geoscience domain for numerical computation and visualization.

This issue is related to the version of Tika implemented to DRAT. DRAT currently uses Tika version 1.6 and that version is unable to detect the MIME type of source codes written in Matlab, causing the mapping operation to perform incorrect classification on Matlab files. Although later versions of Tika (e.g., 1.8) are capable of detecting the correct MIME type of Matlab files (text/x-matlab), not all of the Matlab files can be detected as Matlab file with Tika 1.8 due to a conflict with filename extension (.m) of Objective-C source code files as well with other issues relating to irregular strings at the starting header of Matlab file.

Therefore, we have prepared a custom tika-mimetypes.xml configuration file to improve Tika in terms of detecting Matlab source code. While Tika 1.8 fails to identify 61 out of 103 Matlab files (right of Fig. 6) from the selected 6 packages, Tika 1.8 with the customized configuration file succeeds to correctly detect all the Matlab files (right of Fig. 6). This modification is reflected in the latest released Tika 1.9. We are currently working on upgrading DRAT with the latest version of Tika.

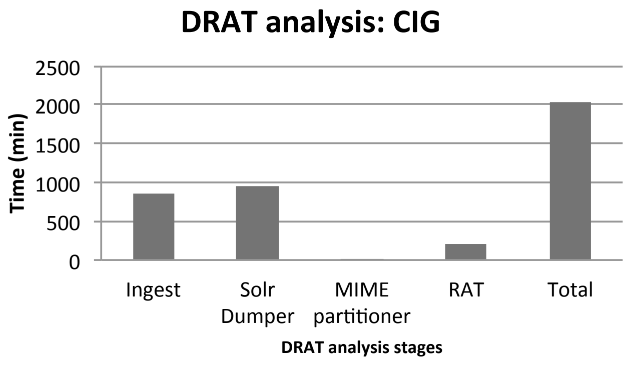


Figure 5. Execution time for DRAT analysis on CIG

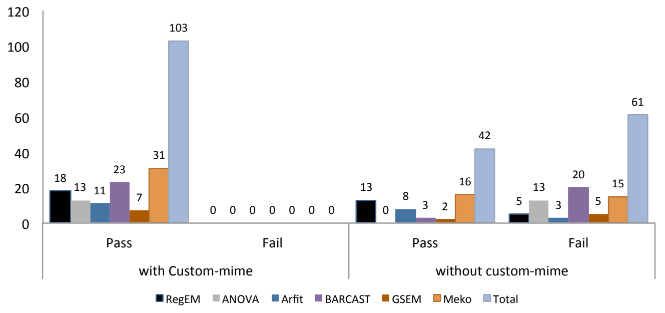


Figure 6. Matlab files detecting test using Tika

V. CONCLUSION AND FUTURE WORK

More so as code is developed in an increasingly open source fashion, we need to run code analysis tools at scale to perform software license checking to make sure software dependencies are using compatible licenses. To this end, DRAT has been developed to bring code auditing and open source license analysis into the realm of *Big Data* using scalable open source Apache technologies. From our early experience, we posit that DRAT will provide an efficient tool for auditing licenses that supports many disciplines, as we have presented the results of a set of applications in the geoscience field.

Our future plan is to improve DRAT to be more applicable for the license verification of source codes produced by scientists from around the world in light of rapidly growing reliance on scientist-created software across scientific fields. In addition, current DRAT only deals with locally available files so we also plan to develop efficient methods to apply DRAT for a remote file system.

We envision DRAT becoming (like RAT) a Maven plugin or a process that fits into continuous integration tools eventually to do license checking. We also envision RAT as not being the only code auditor that we can plug in.

ACKNOWLEDGMENT

This work was partially supported by NSF EarthCube award numbers ICER-1440323 and ICER-1343800. In addition the DARPA XDATA program funded a portion of the work. Effort supported in part by JPL, managed by the California Institute of Technology on behalf of NASA.

REFERENCES

[1] Rosen, L. 2004. Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall.

[2] Open Source Initiative, http://opensource.org, Retrieved: July 2015.

[3] Apache Creadur, http://creadur.apache.org, Retrieved: June 2015.

[4] Apache Incubator, http://incubator.apache.org, Retrieved: June 2015.

[5] Apache Creadur Rat, http://creadur.apache.org/rat/, Retrieved July 2015

[6] DARPA Open Catalog http://opencatalog.darpa.mil, Retrieved: July 2015

[7] Dean, J. and Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. Communications of the ACM 51.1, pp. 107-113. DOI= http://doi.acm.org/10.1145/1327452.1327492

[8] Mattmann, C., Crichton, D., Medvidovic, N., and Hughes, S. 2006. A software architecture-based framework for highly distributed and data intensive scientific applications.In Proceedings of the 28th international conference on Software engineering (Shanghai, China, May 20-28, 2006). ACM, New York, NY, 721-730. DOI= http://doi.acm.org 10.1145/1134285.1134400

[9] Mattmann, C. and Zitting, J. 2011. Tika in Action. Manning Publications Co.

[10] NOAA Paleoclimatology Software Resources https://www.ncdc.noaa.gov/cdo/f?p=517:20:0::::PROXYDATASETLIST:59, Retrieved July 2015

[11] Stodden, V. 2009. Enabling reproducible research: Open licensing for scientific innovation. International Journal of Communications Law and Policy, Forthcoming.

[12] Distributed Release Audit Tool (DRAT), http://github.com/chrismattmann/drat/, Retrieved: July 2015.