

The SWH-Analytics Framework

Alessia Antelmi^{1,3,*}, Massimo Torquati^{2,3}, Daniele Gregori⁴, Francesco Polzella⁵,
Gianmarco Spinatelli⁵ and Marco Aldinucci^{1,3}

¹Computer Science Department, University of Torino

²Computer Science Department, University of Pisa

³HPC-KTT national lab, CINI

⁴E4 Computer Engineering SpA

⁵Zerodivision systems Srl

Abstract

The Software Heritage (SWH) dataset serves as a vast repository for open-source code, with the ambitious goal of preserving all publicly available open-source projects. Despite being designed to effectively archive project files, its size of nearly 1 petabyte presents challenges in efficiently supporting Big Data MapReduce or AI systems. To address this disparity and enable seamless custom analytics on the SWH dataset, we present the SWH-Analytics (SWHA) architecture. This development environment quickly and transparently runs custom analytic applications on open-source software data preserved over time by SWH.

Keywords

Software Heritage, Open-source Software, Large-scale analytics, License management

1. Introduction

At its core definition, open-source software refers to software whose source code is made available to the public via a free and open-source license, which allows viewing, modifying, and distributing the code by anyone at no cost [1]. Over the past twenty years, open-source software has experienced a remarkable evolution, now enjoying extensive adoption. What began as a grassroots movement, marked by the advent of the first freely available open-source operating system, has subsequently evolved into a dominant phenomenon within the developer community [2, 3]. The pivotal role of open source was also highlighted by the 2022 GitHub report [4], which revealed that it serves as the cornerstone of over 90% of the world's software infrastructure.

In this context, the Software Heritage (SWH) initiative represents a valuable source as it aims to archive, preserve, and make accessible all software publicly available in source code form ever produced by humankind [5]. The SWH dataset experiences rapid growth, accumulating several terabytes of data each month. By July 2023, it had reached close to 1 petabyte in size,


ITADATA2023: The 2nd Italian Conference on Big Data and Data Science, September 11–13, 2023, Naples, Italy

*Corresponding author.

✉ alessia.antelmi@unito.it (A. Antelmi); massimo.torquati@unipi.it (M. Torquati);
daniele.gregori@e4company.com (D. Gregori); f.polzella@zerodivision.it (F. Polzella); g.spinatelli@zerodivision.it
(G. Spinatelli); marco.aldinucci@unito.it (M. Aldinucci)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

primarily comprising archived software source code files, each with an average size of less than 4 kilobytes. Simultaneously, the metadata graph associated with this archive had expanded to nearly 20 terabytes. Although being designed to archive and de-duplicate small files, this massive dataset encounters notable difficulties when it comes to effectively serving as input for Big Data processing frameworks like MapReduce (e.g., Spark) or AI systems for training and inference. This challenge arises from the need to navigate through successive files in query results, which may entail traversing the 20-terabyte metadata hash tree and navigating across the vast 1-petabyte storage object repository without any spatial locality.

To bridge the performance gap between stream-based analytics and the SWH dataset, we present the SWH-Analytics (SWHA) architecture. Designed and developed within the context of the ADMIRE European project, the main objective of SWHA is to offer a specialized development and runtime environment tailored for applications aimed at analyzing the extensive repository of open-source software preserved by SWH. A notable feature of SWHA lies in its capability to run custom analytic applications written in Scala. More precisely, in the context of this study, we describe how SWHA could be effectively exploited to investigate license usage in open-source software at a large scale.

The remainder of this paper is organized as follows. Section 2 briefly introduces and describes the SWH dataset. Section 3 details the architecture of SWHA by explaining each component. Section 4 delineates the workflow of a possible application built on top of SWHA. Section 5 concludes this work.

2. Software Heritage

Software Heritage [6, 5, 7] is a globally renowned non-profit initiative established in 2016 with a mission to archive, preserve, and provide access to all publicly available software in its source code form, spanning the entirety of human software production. As of July 2023, this monumental archive encompasses approximately 16.6 billion unique source files and 3.5 billion unique commits from over 258 million development projects, collecting a total of 1 petabyte of data. The SWH repository has been exploited in diverse research endeavours, including examining the geographical and gender diversity in public code contributions [8, 9, 10], analyzing license text variations [11], identifying repository forks [12], and deriving code usage statistics, such as the most commonly used filenames [13], commit patterns [14], and the average size of the most prevalent file types [12].

In SWH, projects are stored as a Merkle directed acyclic graph (DAG) [15]. A Merkle DAG is characterized by a unique identifier for each node, which is derived from the cryptographic properties of the node's content and, in the case of non-leaf nodes, also incorporates the identifiers of their child nodes. This inherent feature of Merkle DAGs endows them with versatility and efficiency, making such structures well-suited for a wide range of applications, including data integrity verification, deduplication, synchronization, and security. The SWH DAG [16] is organized into six logical layers, which represent (i) the raw content of source code files, (ii) project directories, (iii) project revisions or commits, (iv) project releases or tags, (v) project snapshots, and (vi) the project origin.

3. The SWHA infrastructure

The Software Heritage Analytics (SWHA) framework has been designed and developed in the context of the ADMIRE European project¹, whose main objective was to produce software solutions to enhance the throughput of HPC systems and the performance of individual applications. The framework's architecture comprises three primary software layers: storage, data orchestration, and application layers. These layers cooperate in a parallel computing environment managed through the Apache Spark Streaming Framework. A description of each layer follows.

Storage layer. This layer primarily comprises a data cache known as *Cachemire*, designed to enhance the speed of data retrieval. Cachemire is a cache of projects, functioning as a distributed key-value storage system. In this framework, each key corresponds to a unique identifier assigned to a project by SWH, while the associated value contains the project package in a compressed *tgz* format. The Cachemire interface offers a straightforward API that exposes PUT and GET functions, and their implementation relies on the locking mechanisms provided by Posix-compliant file system primitives. Cachemire adopts the LRU algorithm (Least Recently Used) as the cache replacement policy. To efficiently manage the cache size, an external script runs at regular intervals, actively monitoring and ensuring that the size remains within the predefined threshold.

Data Orchestration layer. This layer includes a pool of data stream generators, referred to as *app controllers*, which collaborate with Cachemire in a parallel computing environment managed through the Apache Spark Streaming Framework. More specifically, the orchestration layer uses the official SWH APIs to search for and retrieve projects, which are subsequently processed by Apache Spark workers (as illustrated in Figure 1). Apache Spark is an open-source, distributed computing framework tailored for big data processing and analytics. In this project, we harnessed Spark Streaming to construct low-latency applications, optimizing the time required for data retrieval and computation.

Application layer. SWHA can execute custom analytics applications written in Scala, ensuring compatibility with Apache Spark Streaming. Each application analyzes a specific set of projects defined through a *recipe*. This term is inherited from SWH and is associated with preparing a set of projects for download, informally known as *cooking*. These recipes contain essential information, such as the SWH identifier of the projects to analyze and possibly additional metadata, like the chosen programming language. The application layer acts as the intermediary for communication between an authenticated user and the SWHA system. This interaction occurs through a web-based console accessed using a web browser application. The web console simplifies user interaction with the SWHA system, enabling efficient project searching, application management, and execution within a user-friendly browser environment.

Figure 1 details the execution and data flow within the framework.

¹<https://admire-eurohpc.eu>

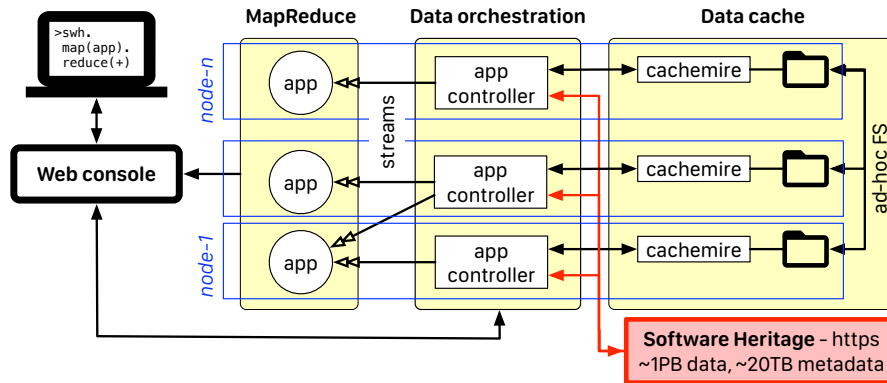


Figure 1: The SWHA data flow.

4. How to exploit SWHA: a case study of the license checker analytic application

An asset of SWHA is that it allows the execution of custom analytic applications written in Scala. Specifically, in this work, we show how SWHA can be effectively exploited to study license inconsistencies potentially across all the revisions of all the source code ever produced or an arbitrary partition of them. Open-source software has gained widespread adoption, with its licensing terms significantly impacting community involvement and contributions [17]. Project licensing, in particular, plays a crucial role in companies, as any violations of licenses can lead to substantial legal risks [18]. With the extensive use of open-source software from public repositories in products, it becomes crucial to strategically understand license mismatches. In this context, the Software Heritage (SWH) dataset is a valuable resource for analyzing project license patterns across temporal and typological dimensions.

The pipeline of a possible application to verify a project’s license(s) compliance comprises three main steps. The initial stage involves providing the application with the designated set of projects a user wishes to examine. This project set is defined using an arbitrarily complex and customizable ‘recipe’ to query the SWH archive via web API. Each *app controller* is responsible for querying the dataset and streaming each project’s files (one file per time) to the analytic application. The second (license identification) and third (license compliance verification) steps represent the core logic and are handled by the custom analytic application. Specifically, for each streamed file, the application looks for an explicit license declaration for the whole project and in-code licenses attached directly to files. The license is automatically detected with ScanCode², one of the most popular open-source license scanners available today. For each project, the application checks whether there is any inconsistency in the licenses detected. If any inconsistency is found, the application verifies whether there is a license conflict by querying the OSADL Open Source License Checklist³, which offers a compatibility matrix between free and open-source (FOSS) licenses.

²<https://github.com/nexB/scancode-toolkit>

³<https://www.osadl.org/OSADL-Open-Source-License-Checklists.oss-compliance-lists.0.html>

The output is a series of statistics about the types of licenses found and whether inconsistencies and conflicts have been detected. Specifically, the application's output may be a JSON file that provides information for each project, including the number and types of licenses detected, their categories, and any inconsistencies or conflicts. Additionally, the application can generate a summary detailing the quantity and types of identified inconsistencies, as well as pairs of licenses causing conflicts.

5. Conclusion

Open-source software is pervasive, and comprehending its development patterns is vital for ensuring the delivery of top-notch software and adequate support to the developer community. In this work, we presented SWHA, a framework designed and developed to offer a specialized development and runtime environment tailored for applications aimed at analyzing the extensive repository of open-source software preserved by SWH. We further described how SWHA could be effectively harnessed to investigate license usage in open-source software, a critical concern due to the legal issues that license violations may lead to. Currently, we are working on implementing such an application. In future work, we aim to assess the performance of the overall SWHA framework, with a particular focus on evaluating the advantages of employing Cachemire as a data cache and comparing the use of one ad hoc file system against another.

Acknowledgments

This work has been partially supported by the EuroHPC JU ADMIRE project (G.A. n. 956748) and the spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by European Union – NextGenerationEU.

References

- [1] RedHat, What is open source software?, <https://www.redhat.com/en/topics/open-source/what-is-open-source-software>, 2022. Accessed on 28/09/2023.
- [2] M.-W. Wu, Y.-D. Lin, Open source software development: an overview, *Computer* 34 (2001) 33–38. doi:10.1109/2.928619.
- [3] F. Bordeleau, P. Meirelles, A. Sillitti, Fifteen years of open source software evolution, in: F. Bordeleau, A. Sillitti, P. Meirelles, V. Lenarduzzi (Eds.), *Open Source Systems*, Springer International Publishing, Cham, 2019, pp. 61–67. doi:10.1007/978-3-030-20883-7_6.
- [4] GitHub, Octoverse 2022: 10 years of tracking open source, <https://github.blog/2022-11-17-octoverse-2022-10-years-of-tracking-open-source>, 2022. Accessed on 28/09/2023.
- [5] R. Di Cosmo, S. Zacchiroli, Software Heritage: Why and How to Preserve Software Source Code, in: *iPRES 2017: 14th International Conference on Digital Preservation*, Kyoto, Japan, 2017.
- [6] R. Di Cosmo, S. Zacchiroli, Software heritage, <https://www.softwareheritage.org>, 2016. Accessed on 28/09/2023.
- [7] J.-F. Abramatic, R. Di Cosmo, S. Zacchiroli, Building the Universal Archive of Source Code, *Communications of the ACM* 61 (2018) 29–31. doi:10.1145/3183558.
- [8] D. Rossi, S. Zacchiroli, Geographic Diversity in Public Code Contributions: An Exploratory Large-Scale Study Over 50 Years, in: *The 2022 Mining Software Repositories Conference (MSR 2022)*, ACM, 2022, pp. 80–85. doi:10.1145/3524842.3528471.

- [9] S. Zacchiroli, Gender differences in public code contributions: a 50-year perspective, *IEEE Software* (2021). doi:10.1109/MS.2020.3038765.
- [10] D. Rossi, S. Zacchiroli, Worldwide Gender Differences in Public Code Contributions (and How They Have Been Affected by the COVID-19 Pandemic), in: 44th International Conference on Software Engineering (ICSE 2022) - Software Engineering in Society (SEIS) Track, ACM, 2022, pp. 172–183. doi:10.1109/ICSE-SEIS55304.2022.9794118.
- [11] S. Zacchiroli, A Large-scale Dataset of (Open Source) License Text Variants, in: The 2022 Mining Software Repositories Conference (MSR 2022), ACM, 2022, pp. 757–761. doi:10.1145/3524842.3528491.
- [12] A. Pietri, G. Rousseau, S. Zacchiroli, Forking without clicking: on how to identify software repository forks, in: MSR 2020: The 17th International Conference on Mining Software Repositories, IEEE, 2020, pp. 277–287. doi:10.1145/3379597.3387450.
- [13] V. Lorentz, R. Di Cosmo, S. Zacchiroli, The Popular Content Filenames Dataset: Deriving Most Likely Filenames from the Software Heritage Archive, 2023. URL: <https://inria.hal.science/hal-04171177>.
- [14] A. Pietri, Organizing the graph of public software development for large-scale mining, Ph.D. thesis, Université Paris Cité, 2021.
- [15] R. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), *Advances in Cryptology – CRYPTO ’87*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 369–378. doi:10.1007/3-540-48184-2_32.
- [16] A. Pietri, D. Spinellis, S. Zacchiroli, The Software Heritage Graph Dataset: Public software development under one roof, in: *Proceedings of the 16th International Conference on Mining Software Repositories, MSR ’19*, IEEE Press, 2019, pp. 138–142. doi:10.1109/MSR.2019.00030.
- [17] J. Gamalielsson, B. Lundell, On licensing and other conditions for contributing to widely used open source projects: An exploratory analysis, in: *Proc. of the 13th Int. Symp. on Open Collaboration, OpenSym ’17*, ACM, NY, USA, 2017. doi:10.1145/3125433.3125456.
- [18] T. Wolter, A. Barcomb, D. Riehle, N. Harutyunyan, Open source license inconsistencies on github, *ACM Trans. Softw. Eng. Methodol.* 32 (2023). doi:10.1145/3571852.